Infrastructure Automation for Containerized Applications on AWS EKS Using Terraform and GitHub Actions

Ayomide Ajayi

larry.ayomide@techbleat.co.uk

Project Overview

This project presents a comprehensive, automated solution for deploying containerized applications on Amazon Elastic Kubernetes Service (EKS) using Infrastructure as Code (IaC) principles. Leveraging Terraform for environment provisioning and GitHub Actions for orchestrating CI/CD pipelines, it delivers a scalable, secure, and production-ready Kubernetes infrastructure.

The deployment architecture includes a custom virtual private cloud (VPC) with three-node EKS clusters spanning two Availability Zones, ensuring high availability and fault tolerance. Infrastructure state management is handled securely using an S3 backend, with DynamoDB providing state locking to support safe, concurrent operations. Declarative Kubernetes manifests are applied post-provisioning to deploy a web application, accessible via an external LoadBalancer service.

The pipeline is structured into modular stages—provisioning, application delivery, and teardown—allowing for repeatable, auditable workflows. A conditional teardown mechanism, activated via manual workflow dispatch (auto_destroy=true), safely decommissions infrastructure after a short delay, supporting both ephemeral and persistent environments. By embracing cloud-native design patterns and automating the full deployment lifecycle, this project advances DevOps maturity, reduces manual overhead, and enhances the reproducibility of infrastructure and application deployment processes.

🛠 Project's Step-by-Step Implementation

1. Push the Image to AWS ECR (*Manually done outside the pipeline but can be integrated if required*)

1.1 Authenticate Docker to ECR:

aws ecr get-login-password --region eu-west-2 | docker login --username AWS --password-stdin <your_aws_account_id>.dkr.ecr.eu-west-2.amazonaws.com

1.2 Create the ECR Repository (if not already created):

aws ecr create-repository --repository-name ghs-nginx-app46 --region eu-west-2

1.3 Tag Your Local Image:

docker tag ghs-nginx-app46:latest < your_aws_account_id >.dkr.ecr.eu-west-2.amazonaws.com/ghs-nginx-app46:latest

1.4 Push the Image to ECR:

docker push < your_aws_account_id >.dkr.ecr.eu-west-2.amazonaws.com/ghs-nginx-app46:latest

2. Initialize Terraform project

- Define provider "aws" with region & credentials.
- Set terraform { backend "s3" { ... } } for remote state in S3, add DynamoDB table for locking. (Backend manually created)
- **Remote state was used for getting VPC outputs via**

  terraform_remote_state

3. Network configuration and provisioning

- Provisioned custom VPC via Terraform, with state managed remotely in an S3 backend for consistency and collaboration.
- Optionally include VPC Endpoint for EKS or S3.

4. EKS cluster provisioning and Managed Node Group

A fully managed EKS control plane and node group are deployed using only Terraform core resources, enabling fine-grained control over IAM roles, subnet associations, and scaling parameters—without relying on third-party modules.

- Create EKS control plane.
- Define worker Node Group: 3 nodes (managed node groups) across private subnets in 2 availability zones (AZs).
- Configure IAM policies and roles amongst others.

5. GitHub Actions CI/CD setup

- Stored Terraform + Kubernetes manifests in a repo.
- Workflow triggers on pushes to main:
    1. Checkout code.
    2. Use hashicorp/setup-terraform.
    3. terraform init, plan, apply (with stored credentials).
    4. Install kubectl && Use aws eks credentials to configure kubectl.
    5. verifies context authentication and node readiness.
    6. Deployed deployment and load balancer service
    7. Post-deployment checks validate LoadBalancer endpoint
    8. Monitor rollout progress to confirm application stability

6. Creation of essential IAM policies including but not limited to:

- Creation of the Node IAM Role
- Creation of an EKS IAM Role for the Control Plane to manage AWS resources.
- Ensure secure GitHub Actions secrets (such as AWS keys).

7. Cost optimisation

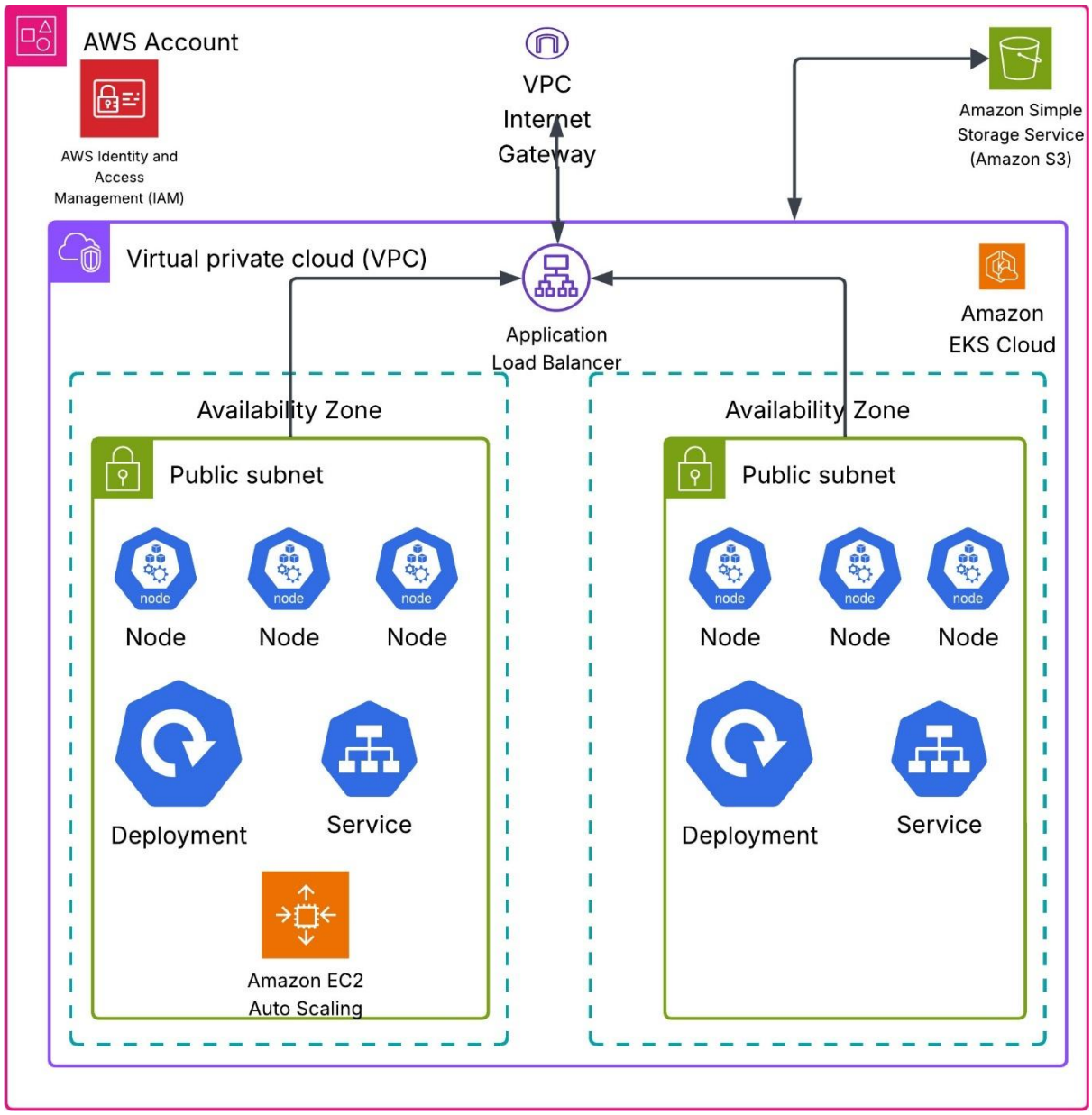
- Implemented a conditional teardown stage gated by a manual workflow_dispatch input (auto_destroy=true).
- Introduces delay before safely destroying the EKS cluster and VPC via Terraform, offering flexibility for ephemeral environments or controlled cleanup.

8. Validation & Monitoring

- Validate cluster health and node automating.
- Confirm app via LoadBalancer endpoint.

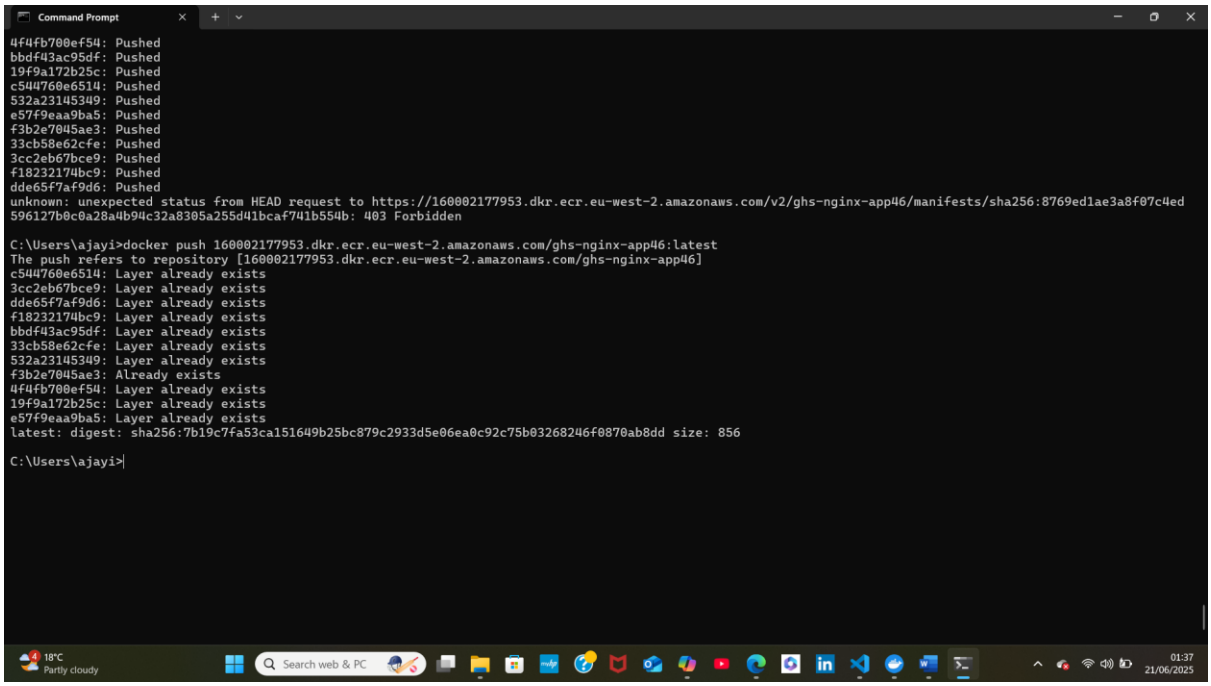
Overall, this project demonstrates proficiency in cloud-native tooling and DevOps principles, highlighting my expertise in AWS IAM, Terraform-based infrastructure-as-code, Kubernetes deployment pipelines, and GitHub Actions for CI/CD automation—all wrapped in a secure, observable, and declarative workflow.

Project Overview Diagram- High-Level Architecture

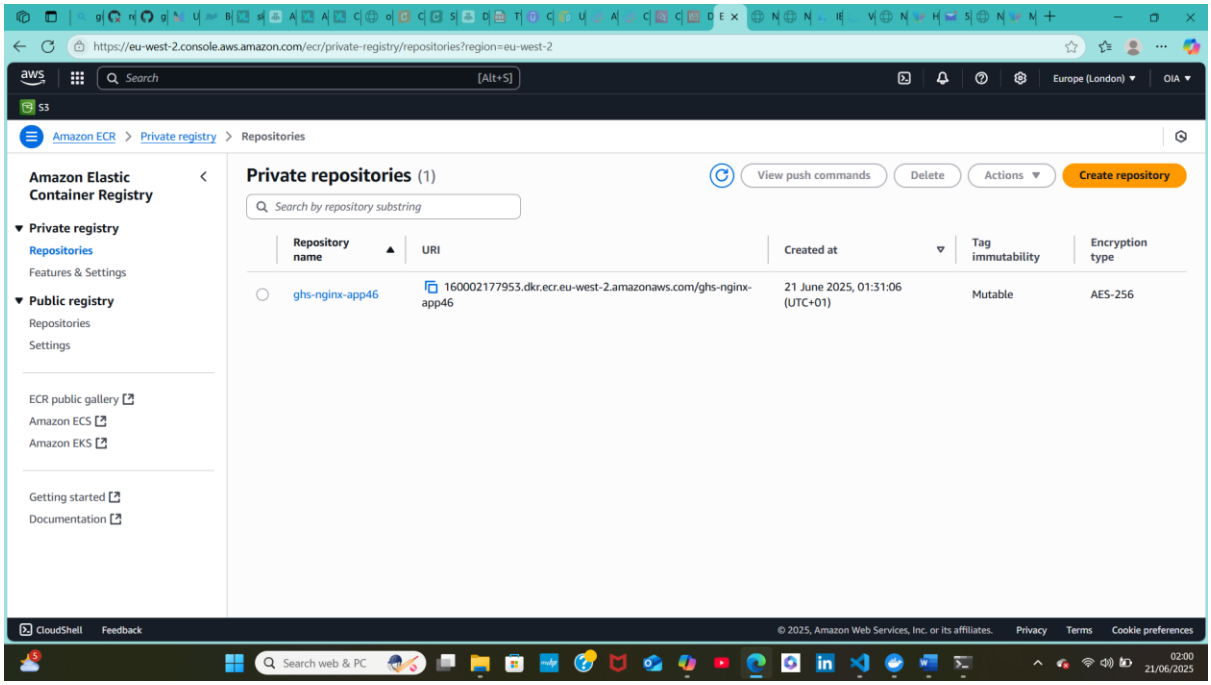


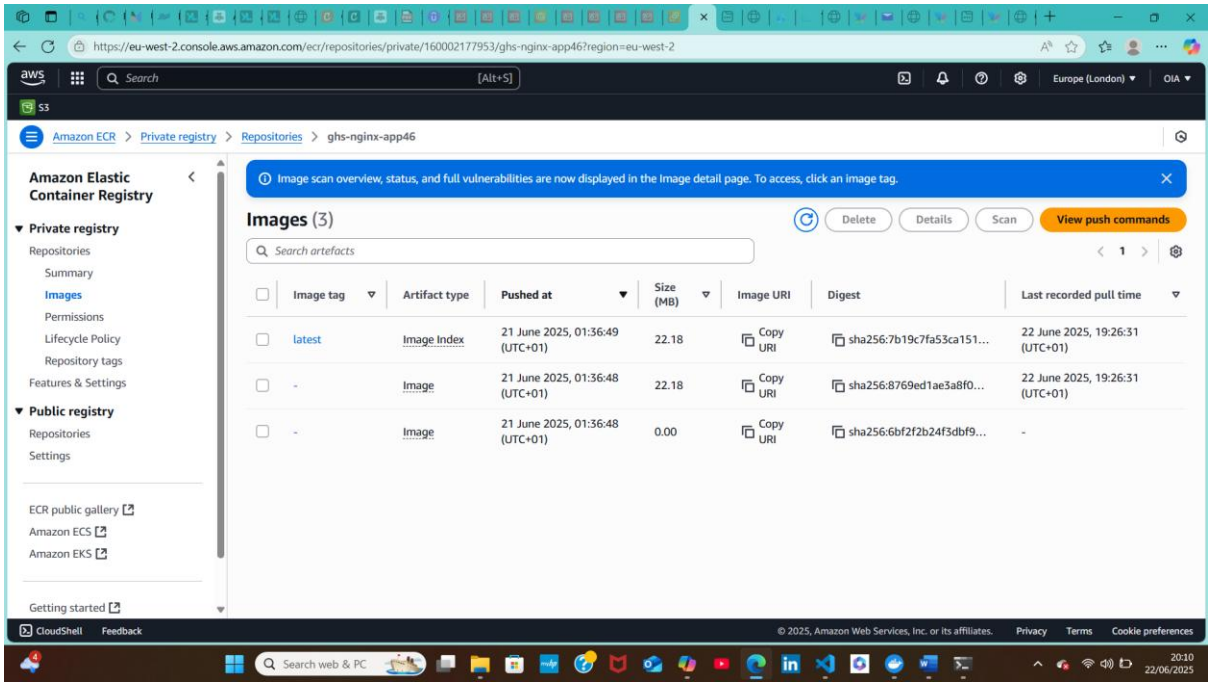
Project Artefacts (a few)

Image Pushed to AWS ECR

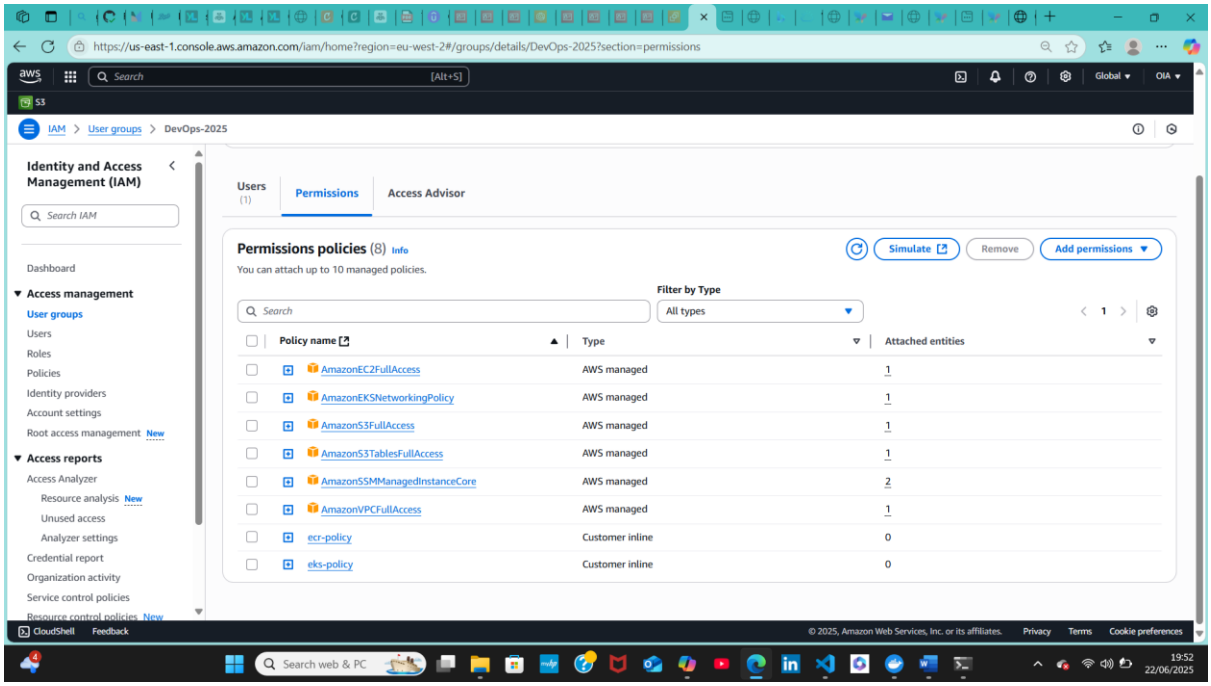


Elastic Container Registry

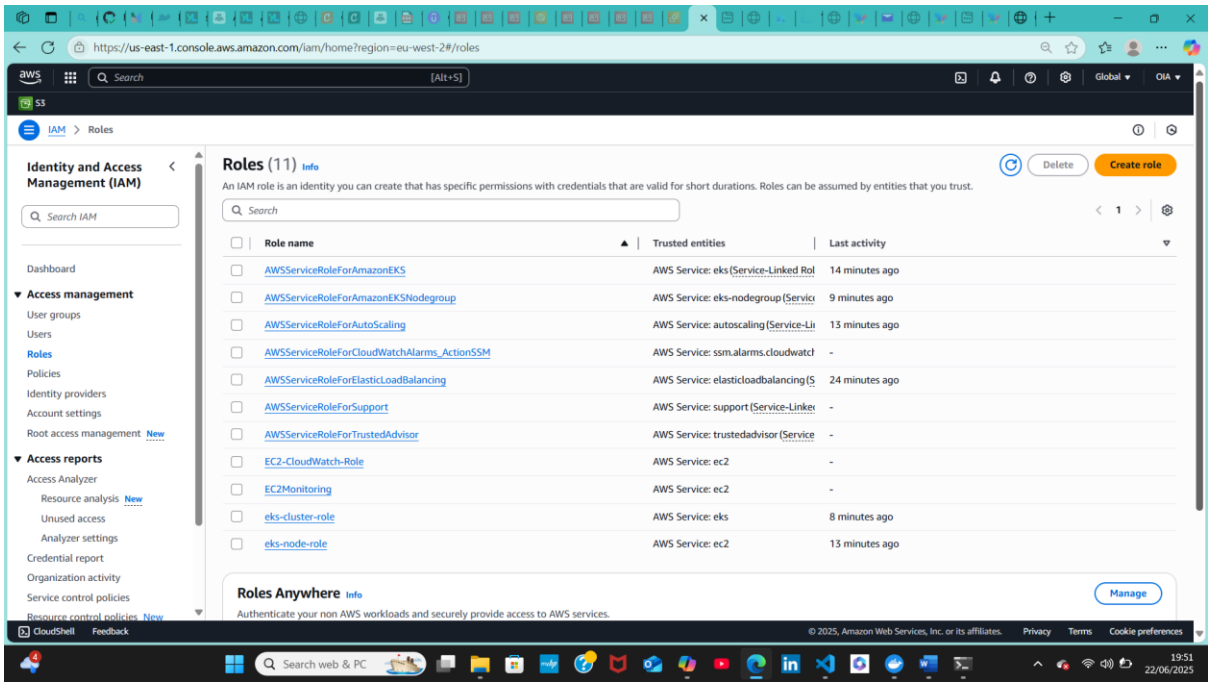




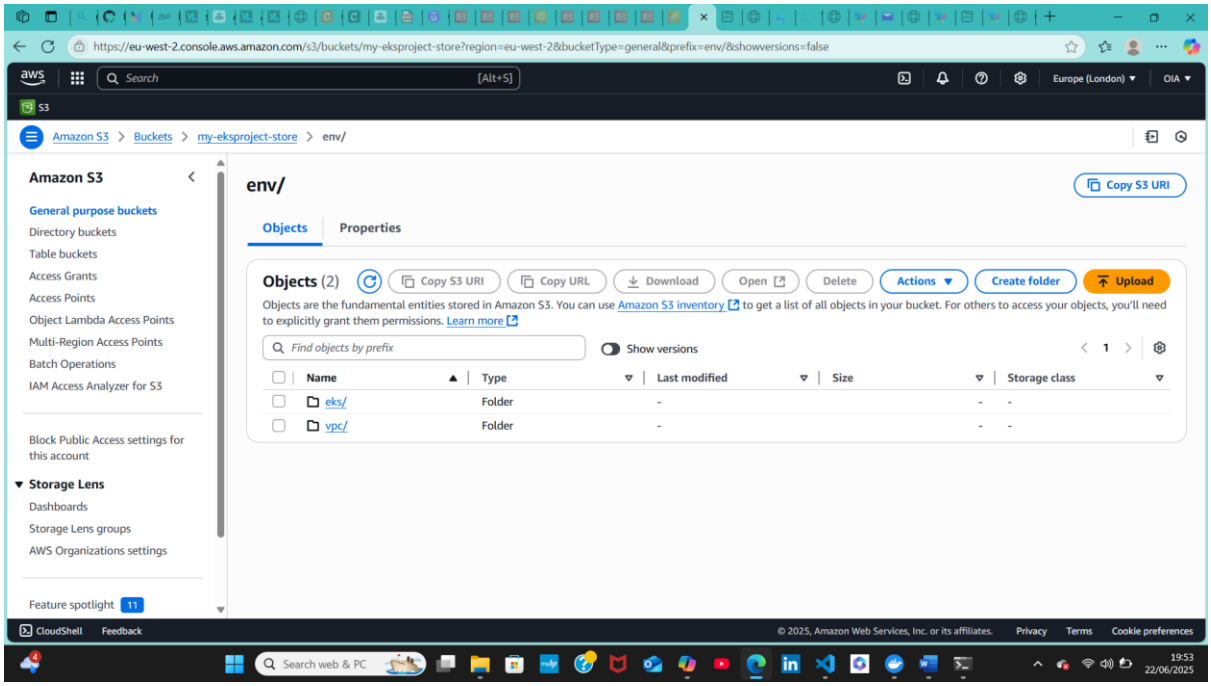
IAM PERMISSIONS

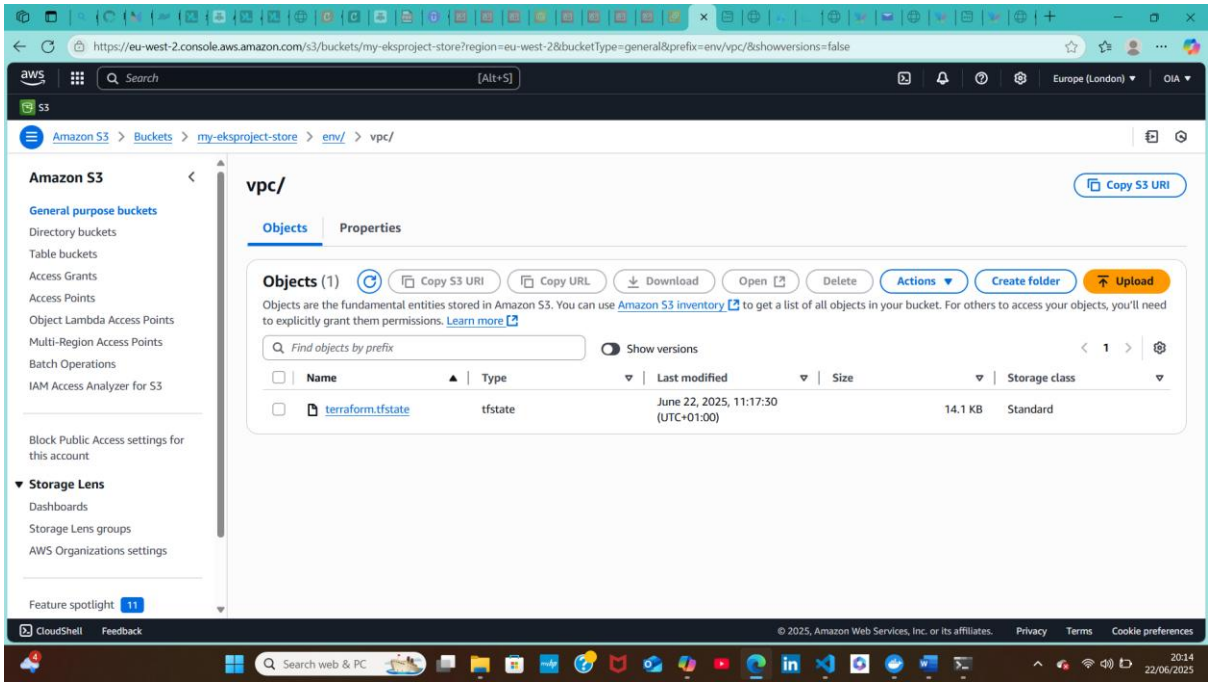


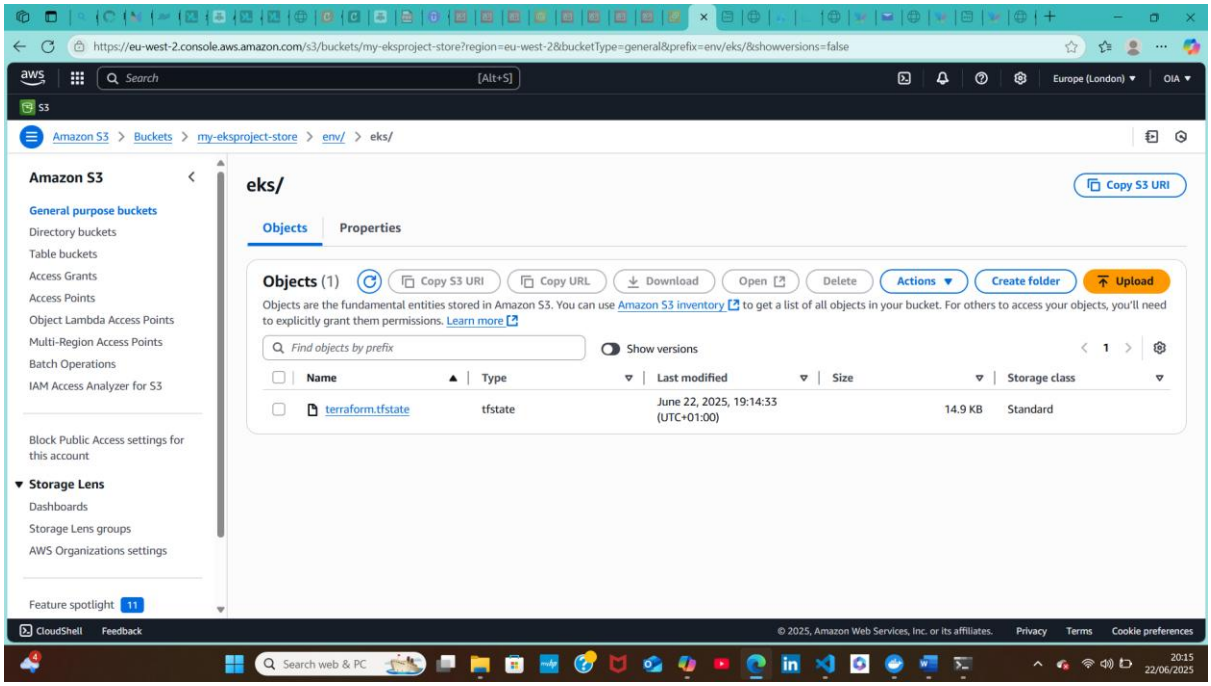
IAM Roles



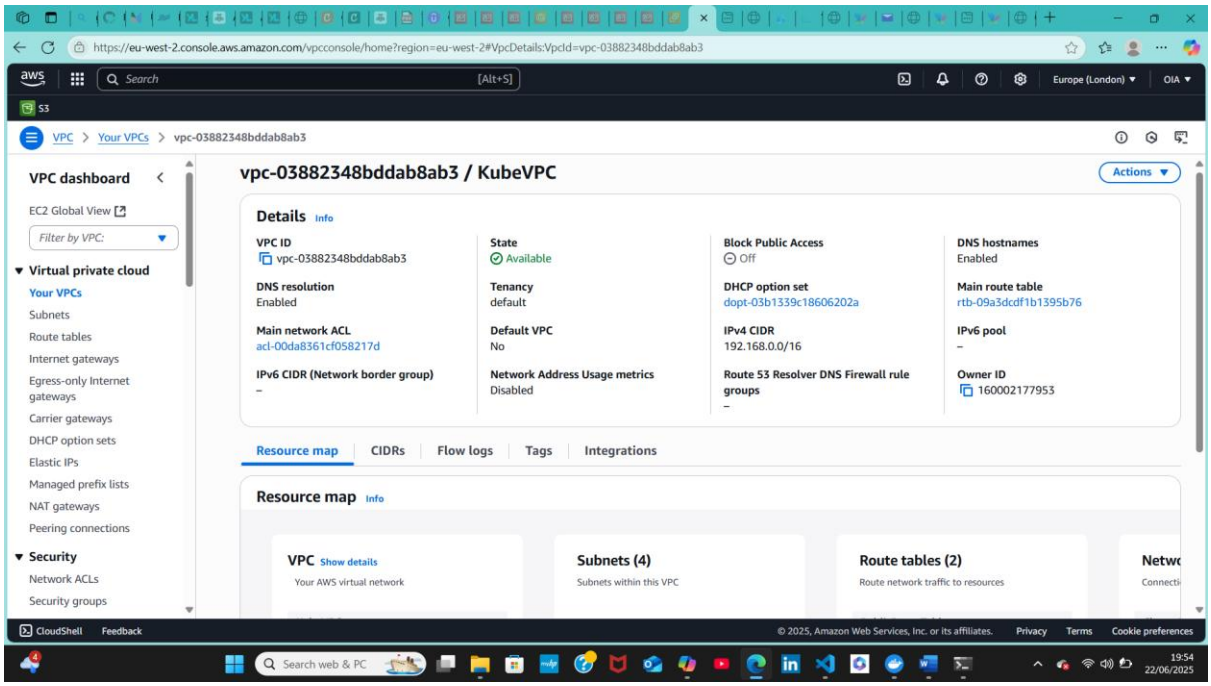
S3 Backend



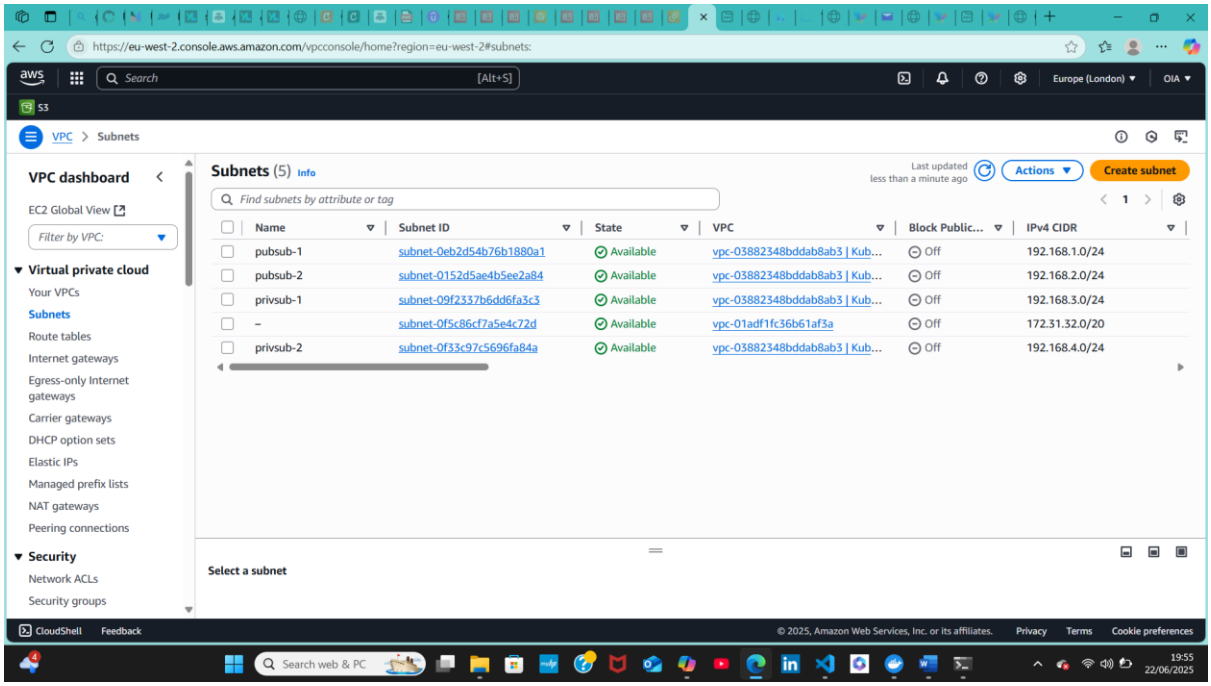

https://eu-west-2.console.aws.amazon.com/s3/buckets/my-eksproject-store?region=eu-west-2&bucketType=general&prefix=env/vpc/&showversions=false
Search [Alt+S]
Europe (London)
OIA
S3
Amazon S3 > Buckets > my-eksproject-store > env/ > vpc/
Amazon S3
General purpose buckets
Directory buckets
Table buckets
Access Grants
Access Points
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
IAM Access Analyzer for S3
Block Public Access settings for this account
Storage Lens
Dashboards
Storage Lens groups
AWS Organizations settings
Feature spotlight 11
vpc/
Copy S3 URI
Objects
Properties
Objects (1)
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more
Find objects by prefix
Show versions
Name
Type
Last modified
Size
Storage class
terraform.tfstate
tfstate
June 22, 2025, 11:17:30 (UTC+01:00)
14.1 KB
Standard
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Search web & PC
20:14
22/06/2025


https://eu-west-2.console.aws.amazon.com/s3/buckets/my-eksproject-store?region=eu-west-2&bucketType=general&prefix=env/eks/&showversions=false
Search [Alt+S]
Europe (London)
OIA
S3
Amazon S3 > Buckets > my-eksproject-store > env/ > eks/
Amazon S3
General purpose buckets
Directory buckets
Table buckets
Access Grants
Access Points
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
IAM Access Analyzer for S3
Block Public Access settings for this account
Storage Lens
Dashboards
Storage Lens groups
AWS Organizations settings
Feature spotlight 11
eks/
Copy S3 URI
Objects
Properties
Objects (1)
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more
Find objects by prefix
Show versions
Name
Type
Last modified
Size
Storage class
terraform.tfstate
tfstate
June 22, 2025, 19:14:33 (UTC+01:00)
14.9 KB
Standard
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Search web & PC
20:15
22/06/2025

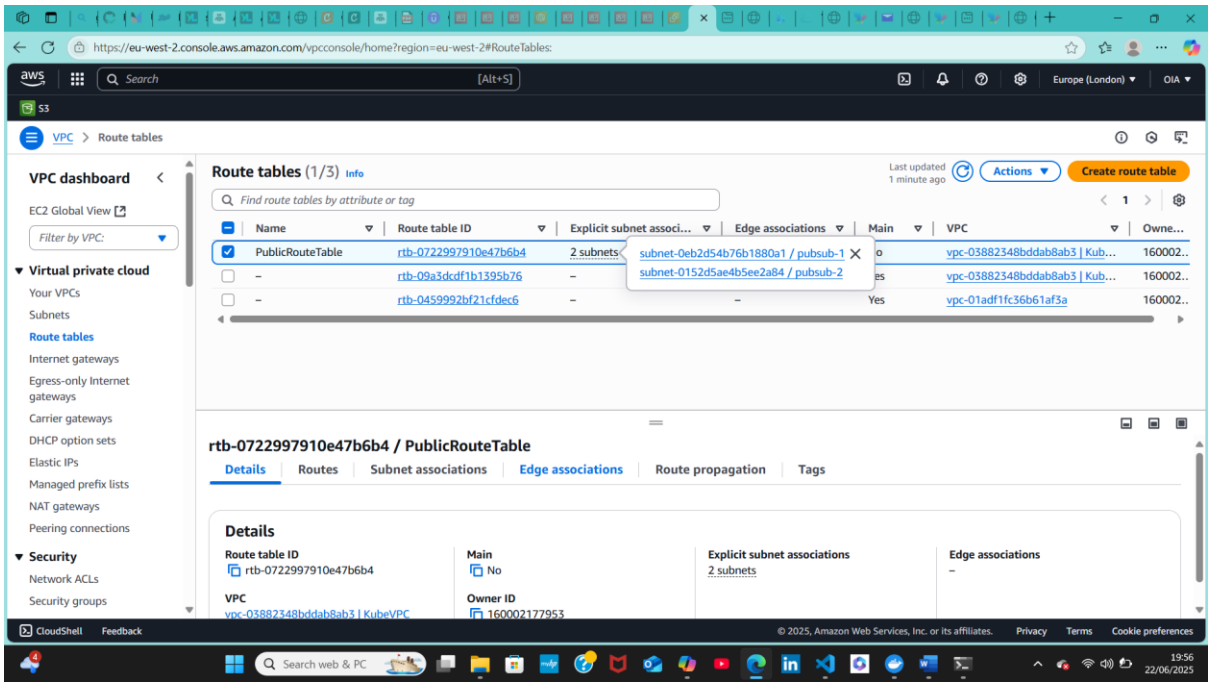
Custom VPC



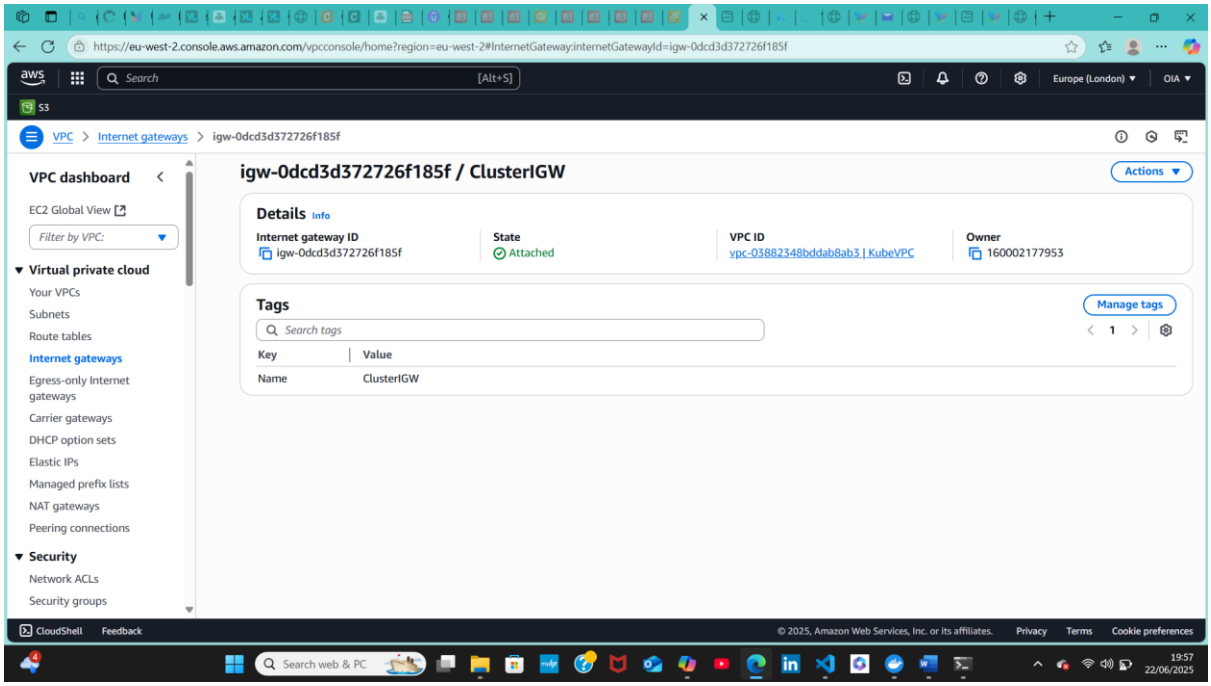
Subnets



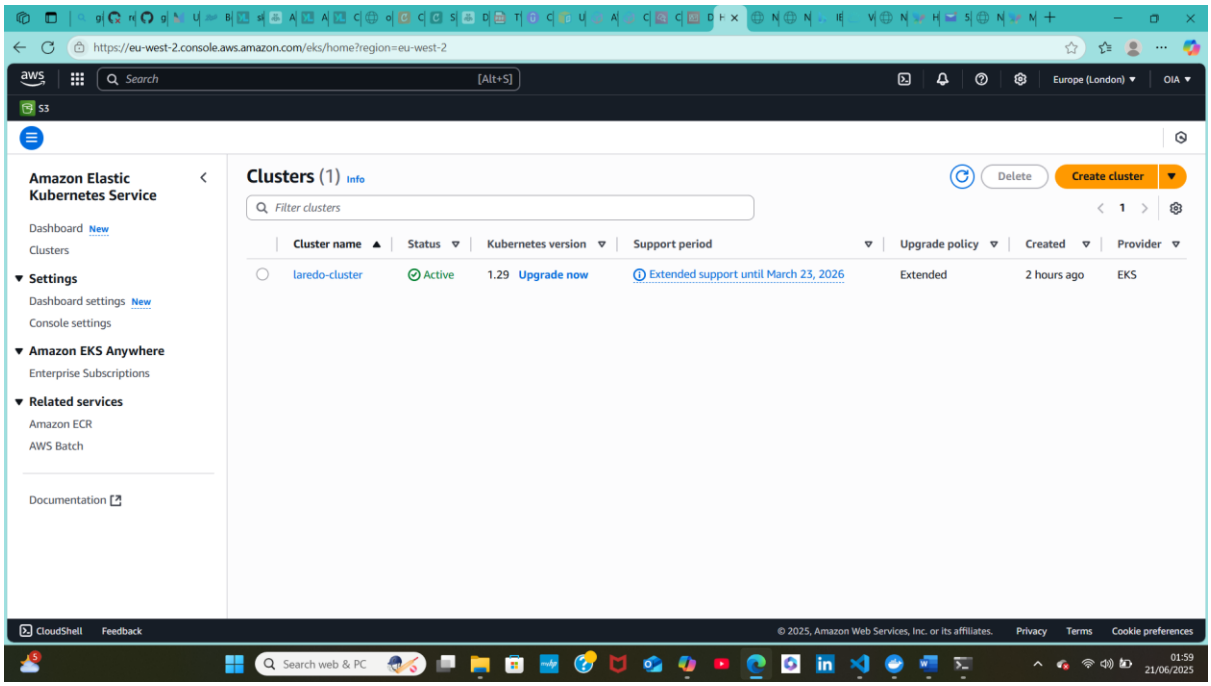
Route tables

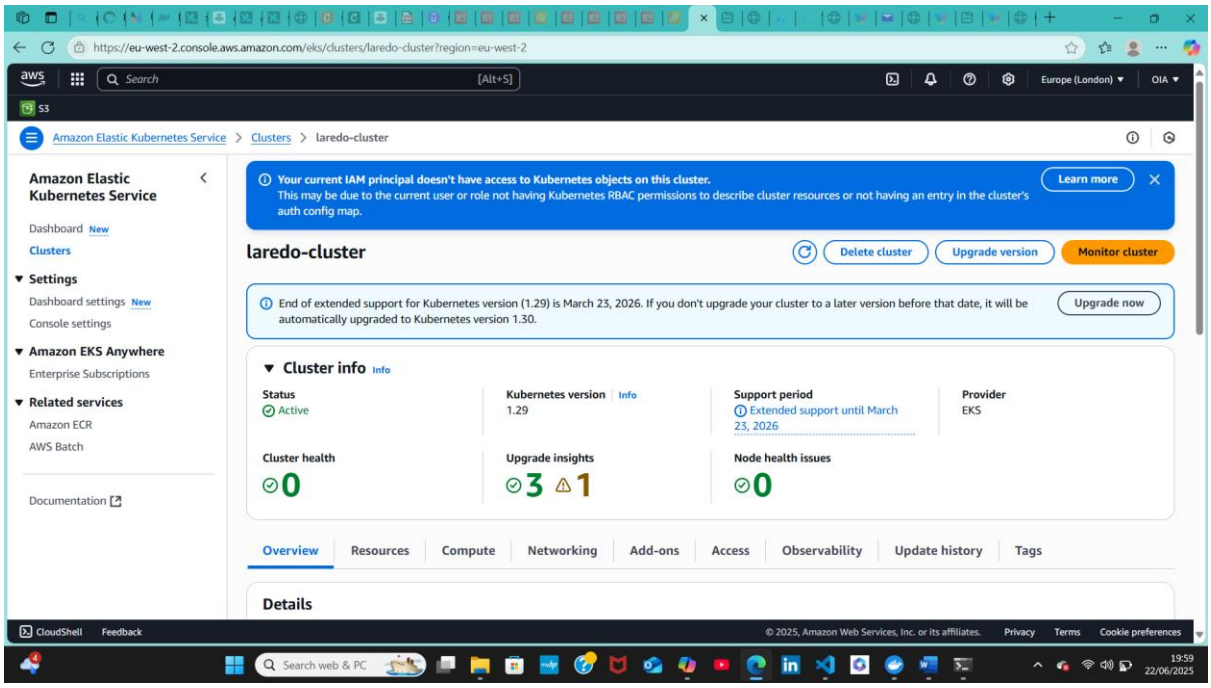


Cluster IGW

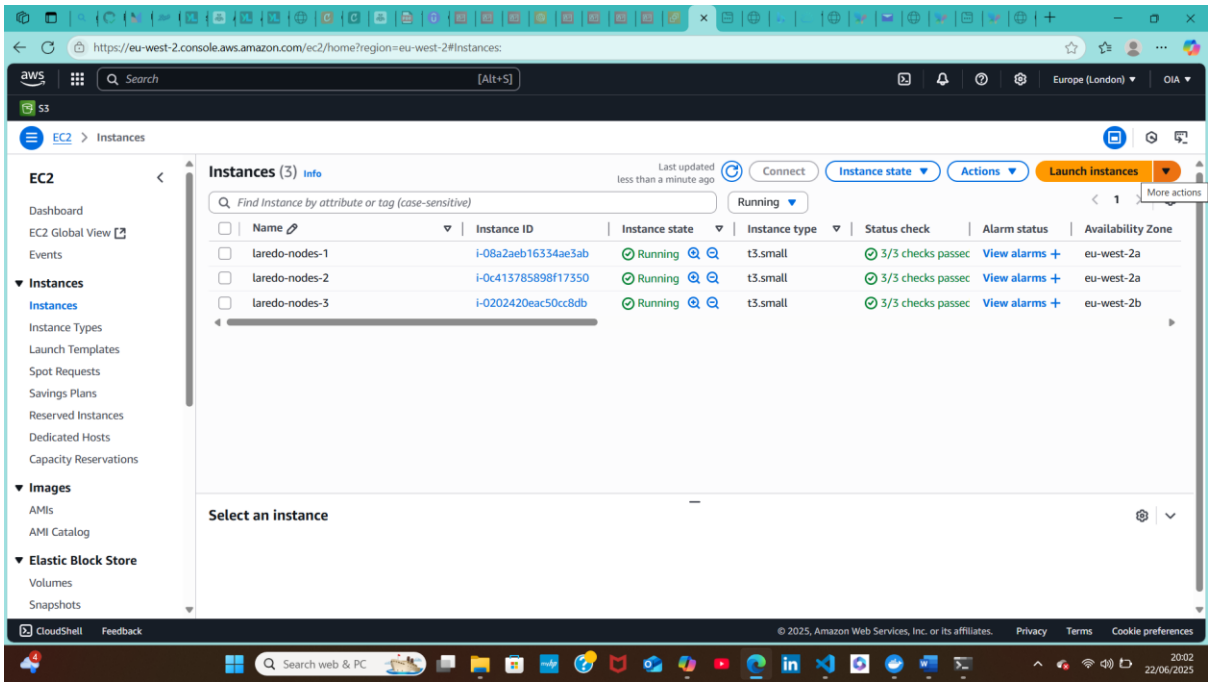


EKS Cluster

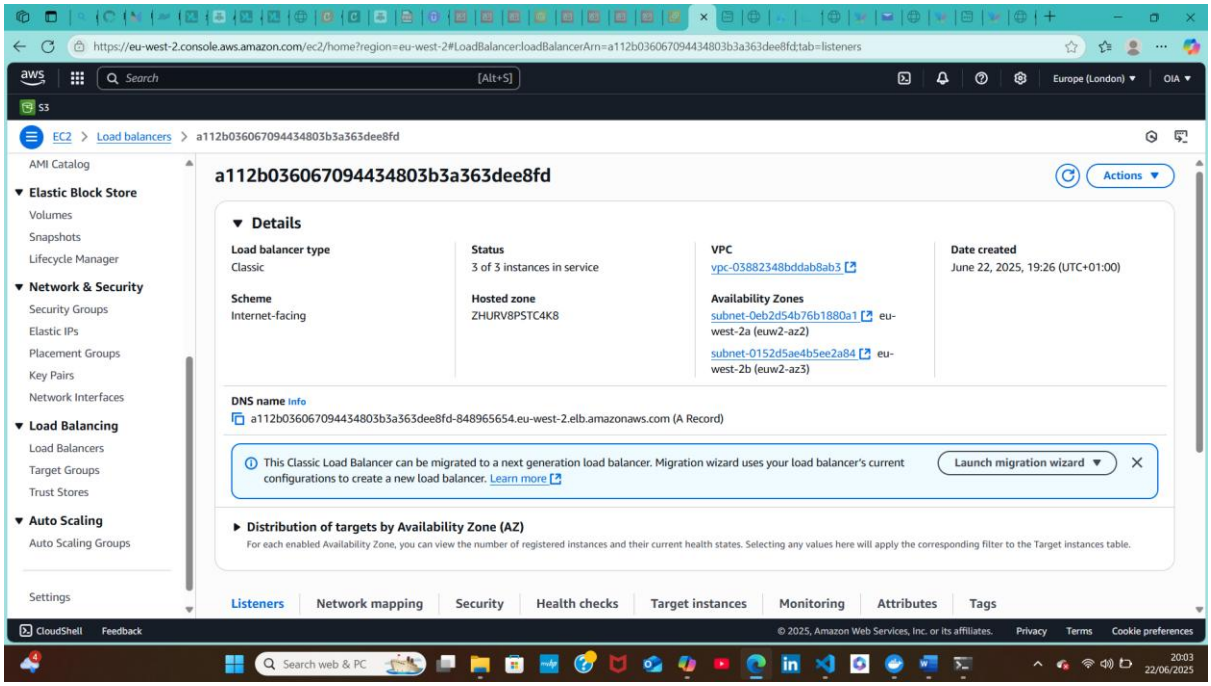

https://eu-west-2.console.aws.amazon.com/eks/home?region=eu-west-2
Amazon Elastic Kubernetes Service
Dashboard New
Clusters
Settings
Dashboard settings New
Console settings
Amazon EKS Anywhere
Enterprise Subscriptions
Related services
Amazon ECR
AWS Batch
Documentation
Clusters (1) Info
Filter clusters
Delete
Create cluster
Cluster name
Status
Kubernetes version
Support period
Upgrade policy
Created
Provider
laredo-cluster
Active
1.29 Upgrade now
Extended support until March 23, 2026
Extended
2 hours ago
EKS
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


https://eu-west-2.console.aws.amazon.com/eks/clusters/laredo-cluster?region=eu-west-2
Amazon Elastic Kubernetes Service > Clusters > laredo-cluster
Your current IAM principal doesn't have access to Kubernetes objects on this cluster.
This may be due to the current user or role not having Kubernetes RBAC permissions to describe cluster resources or not having an entry in the cluster's auth config map.
Learn more
laredo-cluster
Delete cluster
Upgrade version
Monitor cluster
End of extended support for Kubernetes version (1.29) is March 23, 2026. If you don't upgrade your cluster to a later version before that date, it will be automatically upgraded to Kubernetes version 1.30.
Upgrade now
Cluster info Info
Status
Active
Kubernetes version Info
1.29
Support period
Extended support until March 23, 2026
Provider
EKS
Cluster health
0
Upgrade insights
3 1
Node health issues
0
Overview
Resources
Compute
Networking
Add-ons
Access
Observability
Update history
Tags
Details

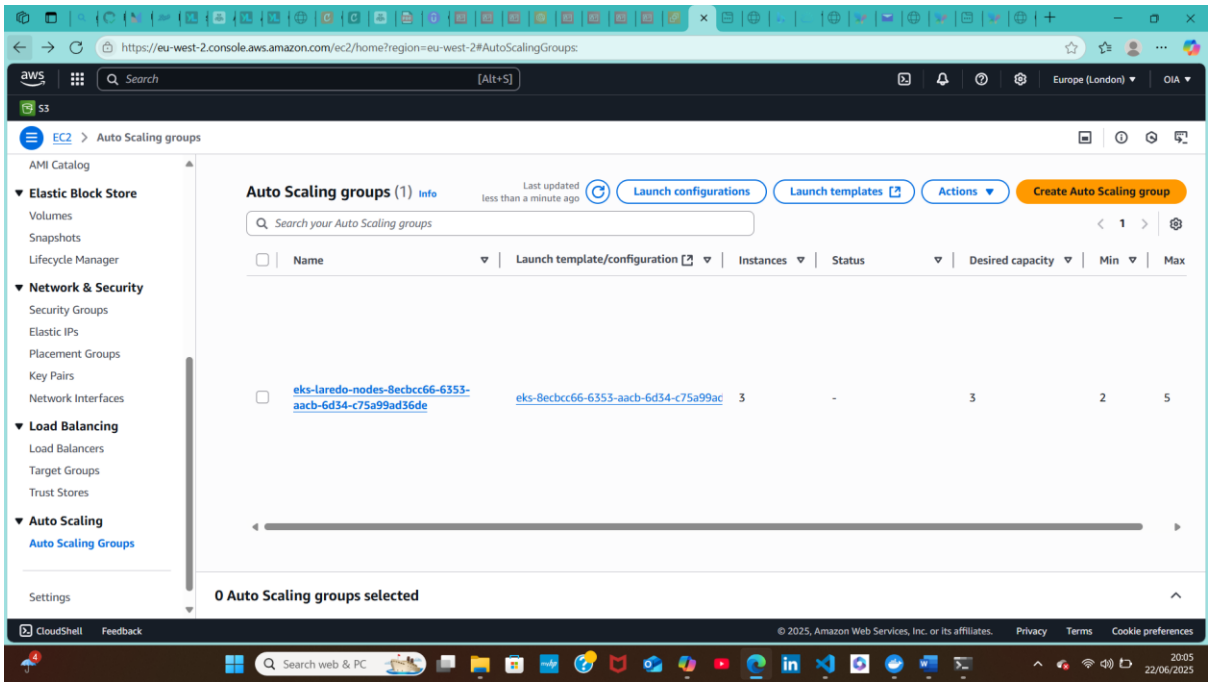
NodeGroup

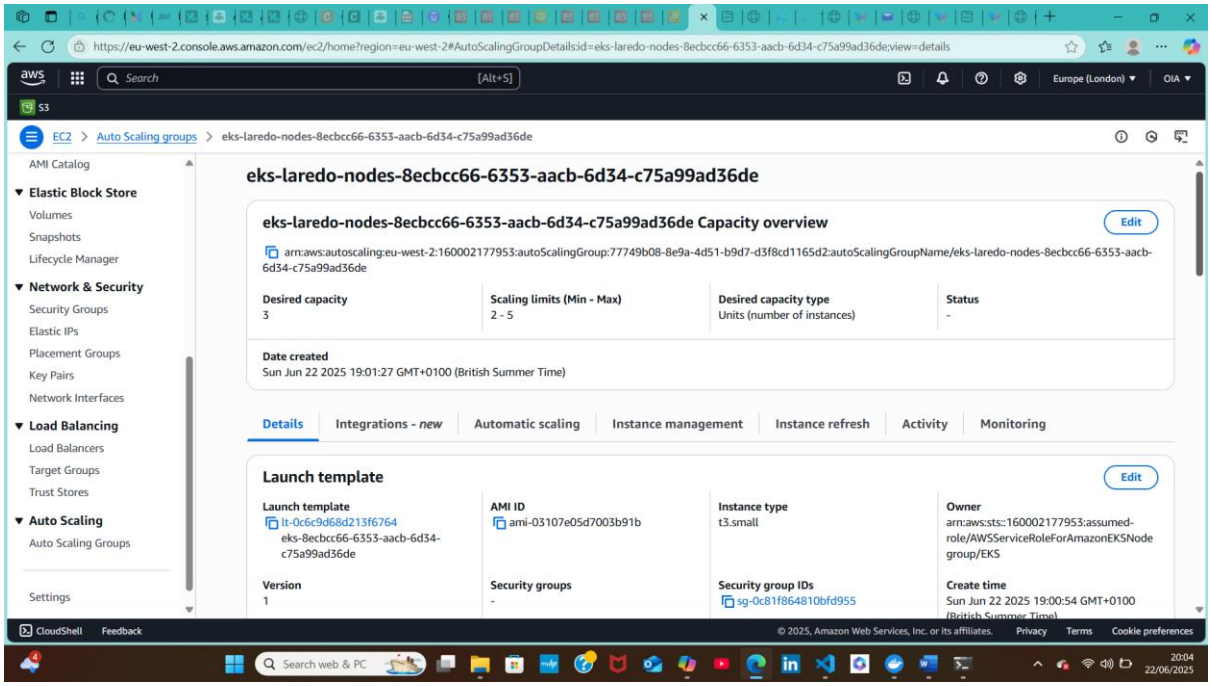


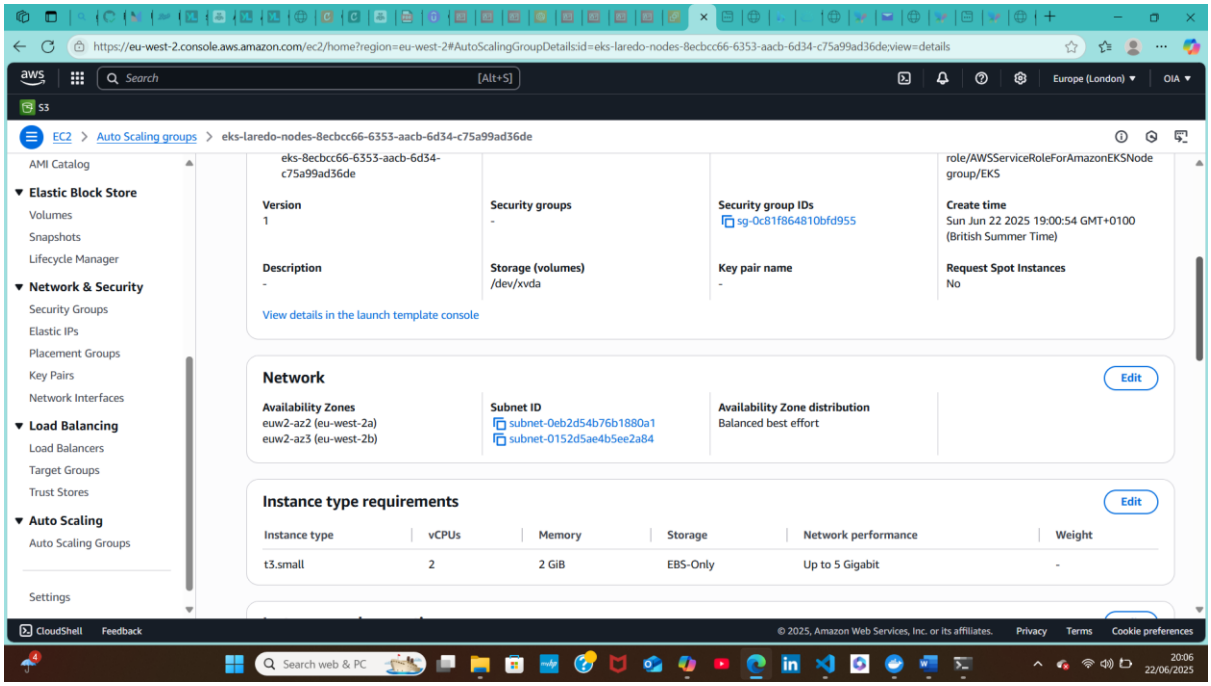
Load Balancer

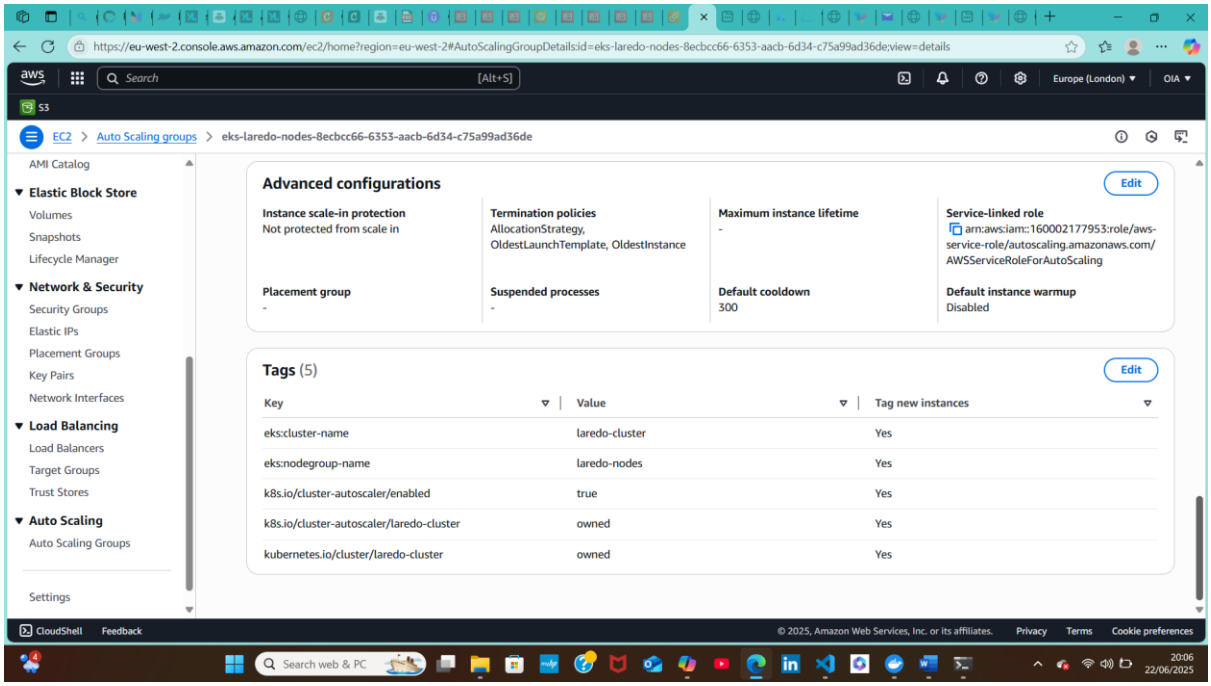


Auto Scaling Group (ASG)





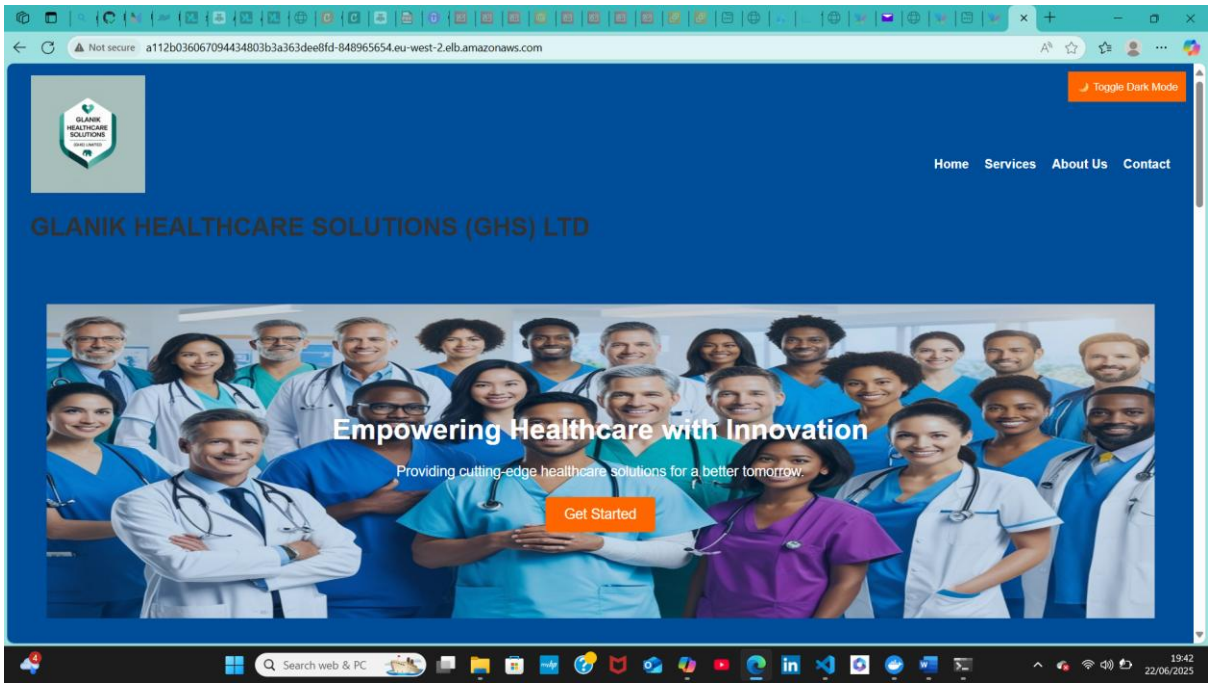

https://eu-west-2.console.aws.amazon.com/ec2/home?region=eu-west-2#AutoScalingGroupDetails:id=eks-laredo-nodes-8ecbcc66-6353-aacb-6d34-c75a99ad36de;view=details
Search [Alt+S]
Europe (London)
OIA
S3
EC2 > Auto Scaling groups > eks-laredo-nodes-8ecbcc66-6353-aacb-6d34-c75a99ad36de
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Network & Security
Security Groups
Elastic IPs
Placement Groups
Key Pairs
Network Interfaces
Load Balancing
Load Balancers
Target Groups
Trust Stores
Auto Scaling
Auto Scaling Groups
Settings
eks-8ecbcc66-6353-aacb-6d34-c75a99ad36de
Version
1
Security groups
-
Security group IDs
sg-0c81f864810bfd955
Create time
Sun Jun 22 2025 19:00:54 GMT+0100 (British Summer Time)
role/AWSServiceRoleForAmazonEKSNodegroup/EKS
Description
-
Storage (volumes)
/dev/xvda
Key pair name
-
Request Spot Instances
No
View details in the launch template console
Network
Edit
Availability Zones
euw2-az2 (eu-west-2a)
euw2-az3 (eu-west-2b)
Subnet ID
subnet-0eb2d54b76b1880a1
subnet-0152d5ae4b5ee2a84
Availability Zone distribution
Balanced best effort
Instance type requirements
Edit
Instance type
vCPUs
Memory
Storage
Network performance
Weight
t3.small
2
2 GiB
EBS-Only
Up to 5 Gigabit
-
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Search web & PC
20:06
22/06/2025

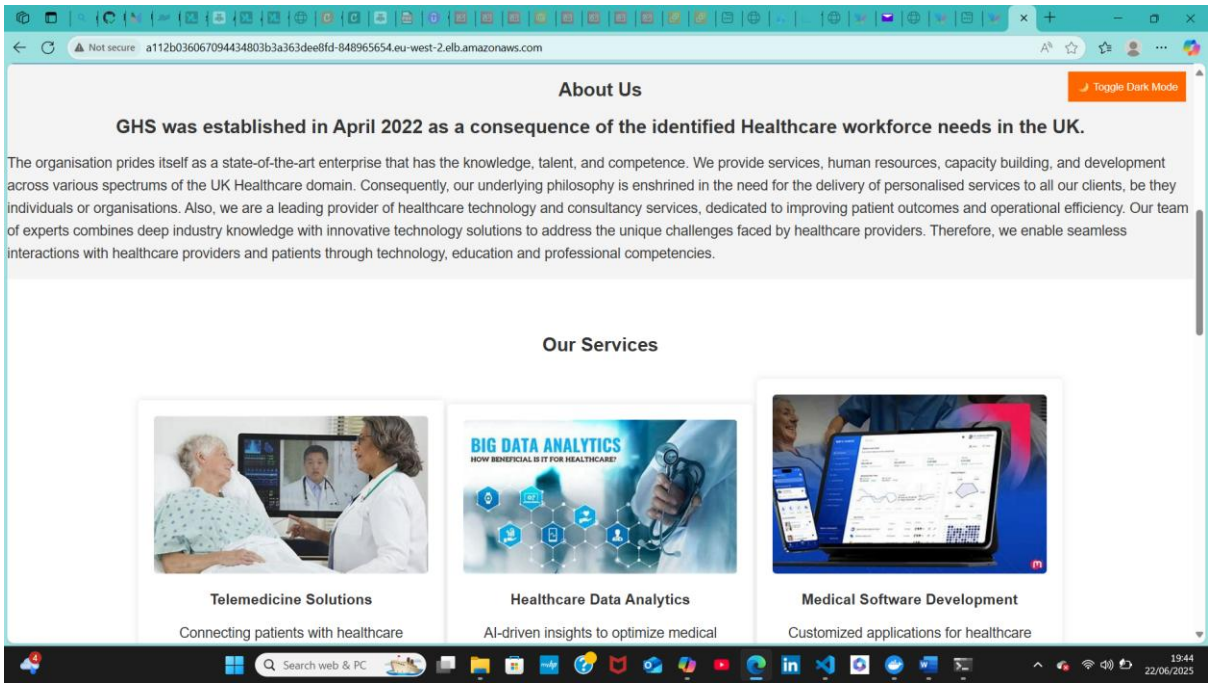

https://eu-west-2.console.aws.amazon.com/ec2/home?region=eu-west-2#AutoScalingGroupDetails:id=eks-laredo-nodes-8ecbcc66-6353-aacb-6d34-c75a99ad36de;view=details
Search [Alt+S]
Europe (London)
OIA
S3
EC2 > Auto Scaling groups > eks-laredo-nodes-8ecbcc66-6353-aacb-6d34-c75a99ad36de
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Network & Security
Security Groups
Elastic IPs
Placement Groups
Key Pairs
Network Interfaces
Load Balancing
Load Balancers
Target Groups
Trust Stores
Auto Scaling
Auto Scaling Groups
Settings
Advanced configurations
Edit
Instance scale-in protection
Not protected from scale in
Termination policies
AllocationStrategy, OldestLaunchTemplate, OldestInstance
Maximum instance lifetime
-
Service-linked role
arn:aws:iam::160002177953:role/aws-service-role/autoscaling.amazonaws.com/AWSServiceRoleForAutoScaling
Placement group
-
Suspended processes
-
Default cooldown
300
Default instance warmup
Disabled
Tags (5)
Edit
Key
Value
Tag new instances
eks:cluster-name
laredo-cluster
Yes
eks:nodegroup-name
laredo-nodes
Yes
k8s.io/cluster-autoscaler/enabled
true
Yes
k8s.io/cluster-autoscaler/laredo-cluster
owned
Yes
kubernetes.io/cluster/laredo-cluster
owned
Yes
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Search web & PC
20:06
22/06/2025

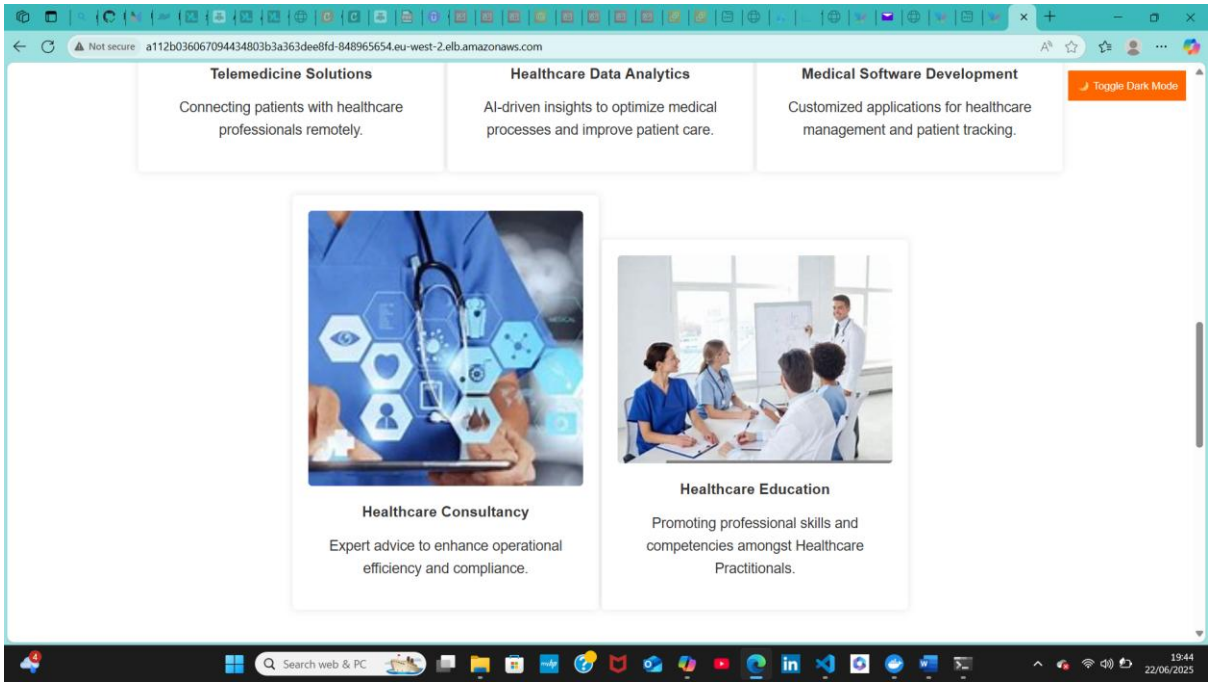
Loadbalancer Endpoint

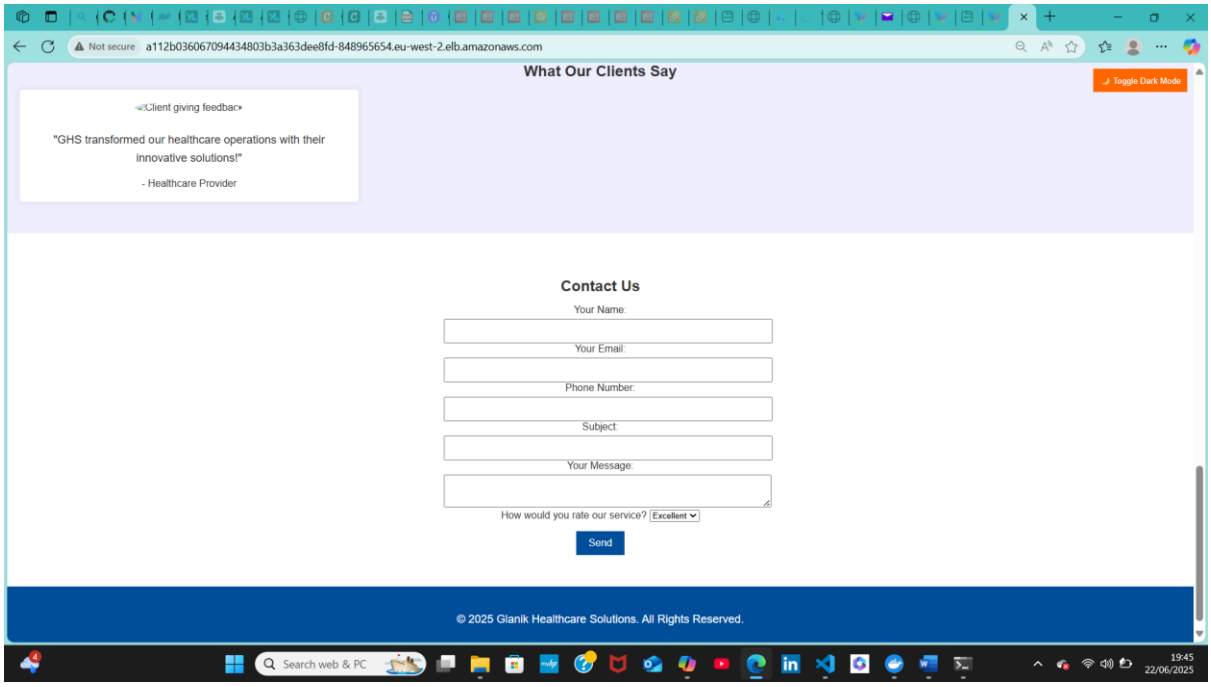
LoadBalancer endpoint is available at: http://a112b036067094434803b3a363dee8fd-848965654.eu-west-2.elb.amazonaws.com

Web application Access via LoadBalancer Service Using the Endpoint above

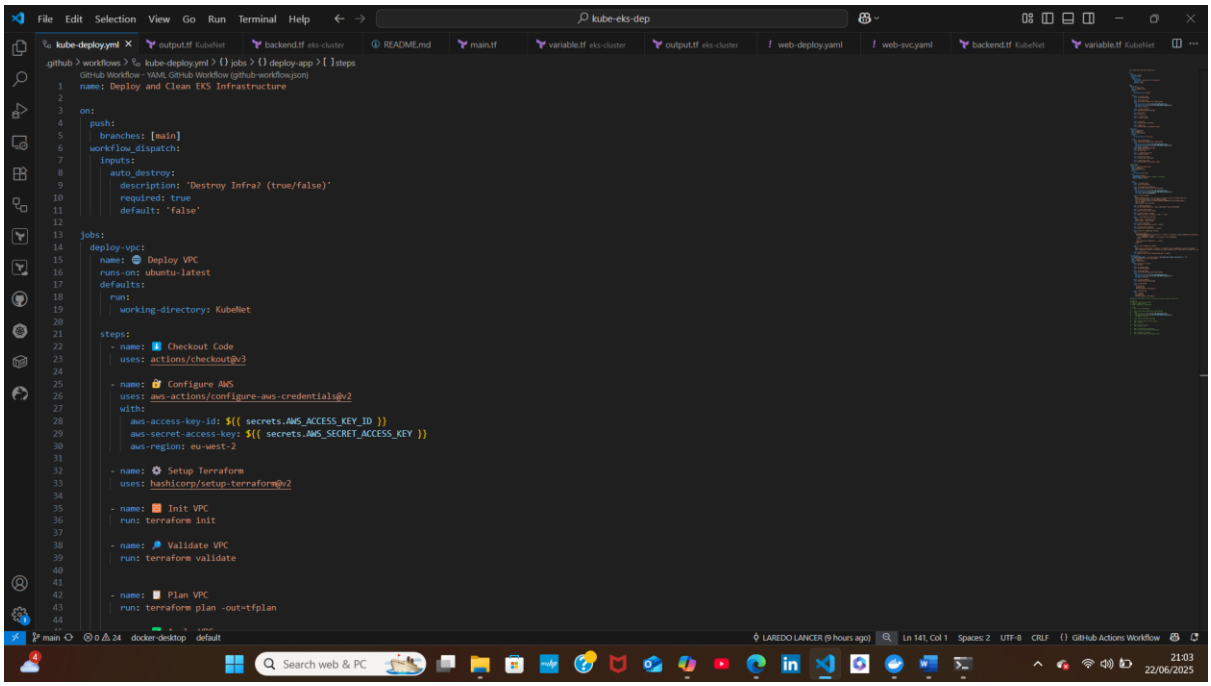




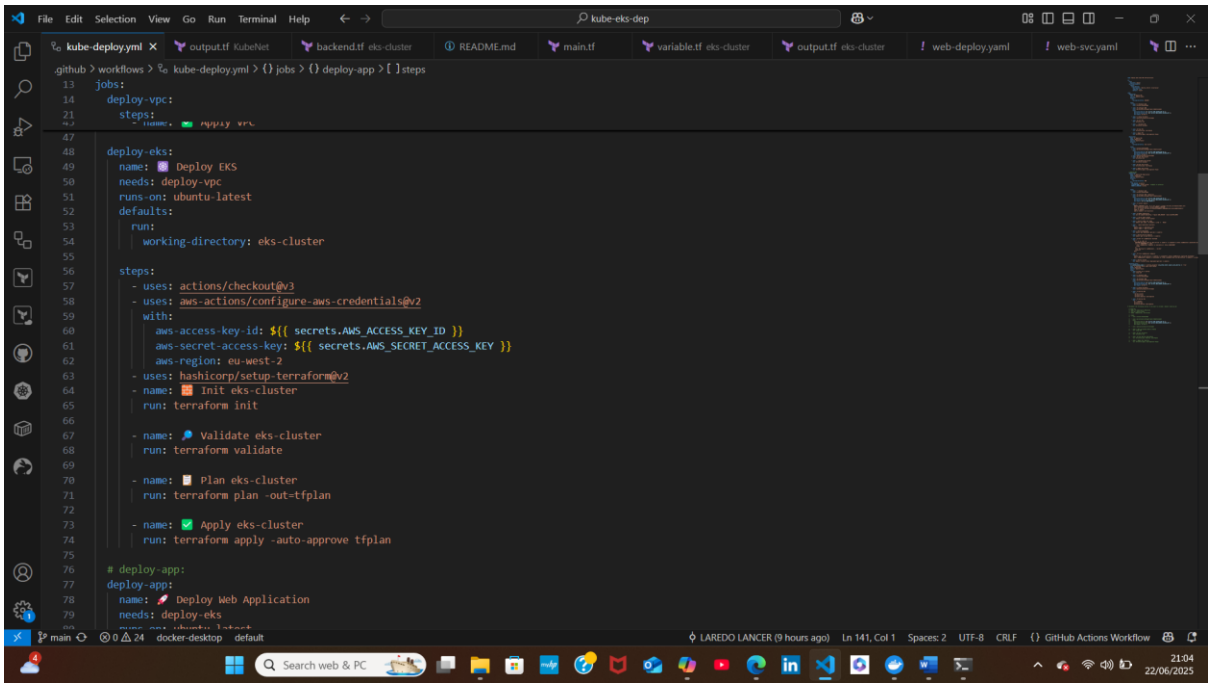

Not secure
a112b0360670944348O3b3a363dee8fd-848965654.eu-west-2.elb.amazonaws.com
Telemedicine Solutions
Connecting patients with healthcare professionals remotely.
Healthcare Data Analytics
AI-driven insights to optimize medical processes and improve patient care.
Medical Software Development
Customized applications for healthcare management and patient tracking.
Toggle Dark Mode
Healthcare Consultancy
Expert advice to enhance operational efficiency and compliance.
Healthcare Education
Promoting professional skills and competencies amongst Healthcare Practitionals.
Search web & PC
19:44
22/06/2025


Not secure
a112b0360670944348O3b3a363dee8fd-848965654.eu-west-2.elb.amazonaws.com
What Our Clients Say
Toggle Dark Mode
Client giving feedback
"GHS transformed our healthcare operations with their innovative solutions!"
- Healthcare Provider
Contact Us
Your Name:
Your Email:
Phone Number:
Subject:
Your Message:
How would you rate our service? Excellent
Send
© 2025 Gianik Healthcare Solutions. All Rights Reserved.
Search web & PC
19:45
22/06/2025

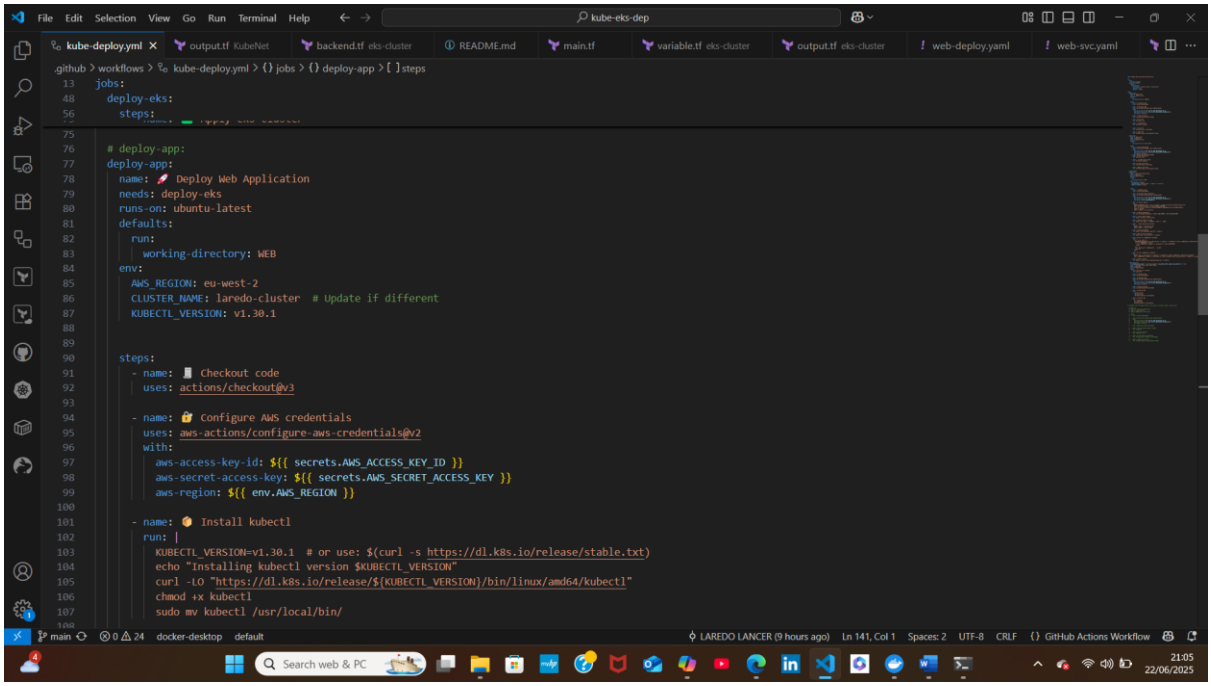
GitHub Actions Workflow



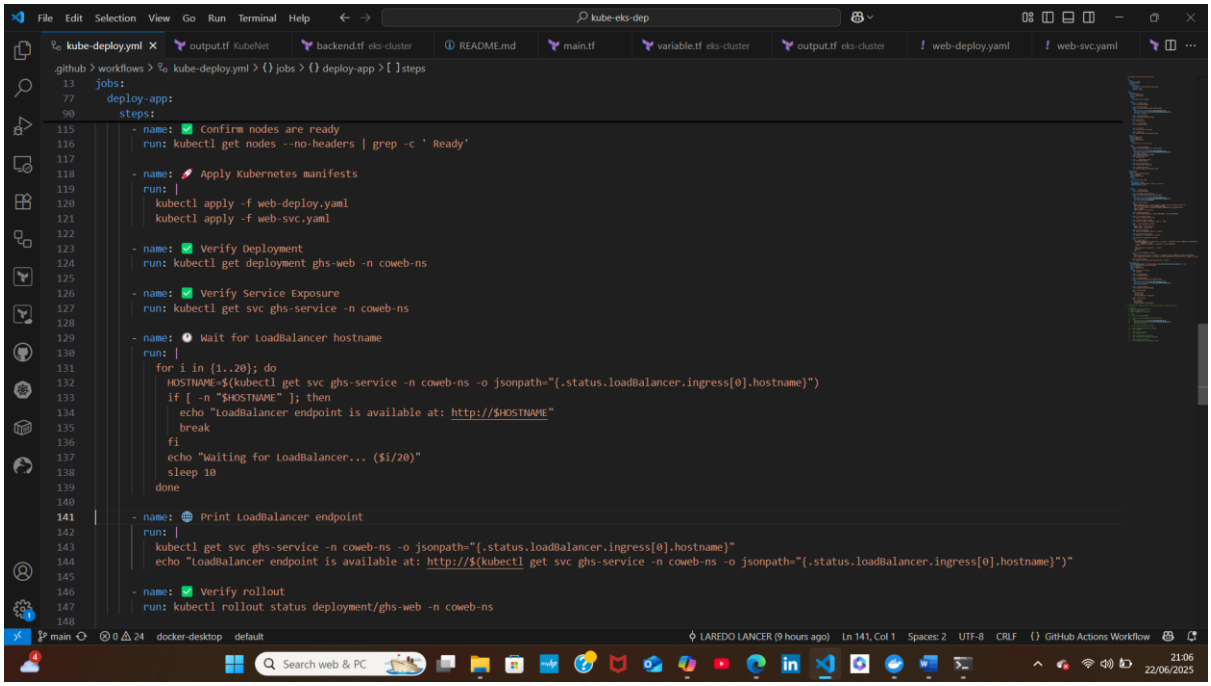
```yaml
name: Deploy and Clean EKS Infrastructure

on:
  push:
    branches: [main]
  workflow_dispatch:
    inputs:
      auto_destroy:
        description: 'Destroy Infra? (true/false)'
        required: true
        default: 'false'

jobs:
  deploy-vpc:
    name: 🌐 Deploy VPC
    runs-on: ubuntu-latest
    defaults:
      run:
        working-directory: KubeNet

    steps:
      - name: 📥 Checkout Code
        uses: actions/checkout@v3

      - name: 🔐 Configure AWS
        uses: aws-actions/configure-aws-credentials@v2
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: eu-west-2

      - name: ⚙️ Setup Terraform
        uses: hashicorp/setup-terraform@v2

      - name: 🧱 Init VPC
        run: terraform init

      - name: 🔎 Validate VPC
        run: terraform validate


      - name: 📋 Plan VPC
        run: terraform plan -out=tfplan
```



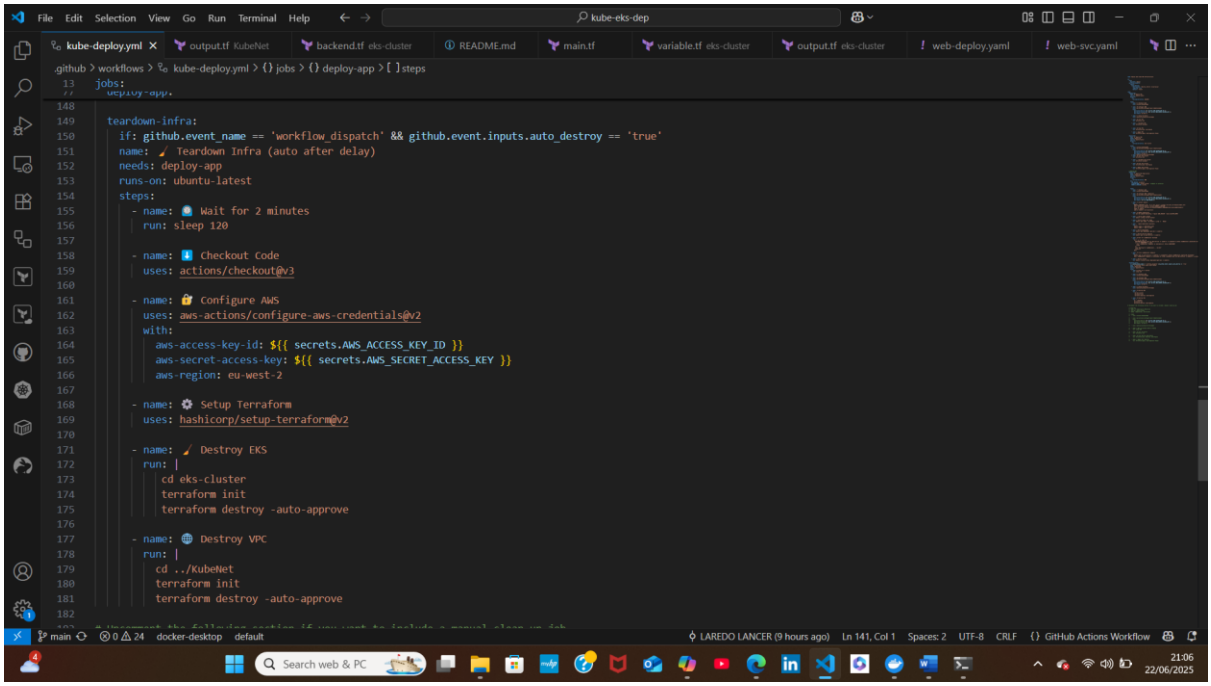
```yaml
  deploy-eks:
    name: ☸️ Deploy EKS
    needs: deploy-vpc
    runs-on: ubuntu-latest
    defaults:
      run:
        working-directory: eks-cluster

    steps:
      - uses: actions/checkout@v3
      - uses: aws-actions/configure-aws-credentials@v2
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: eu-west-2
      - uses: hashicorp/setup-terraform@v2
      - name: 🧱 Init eks-cluster
        run: terraform init

      - name: 🔎 Validate eks-cluster
        run: terraform validate

      - name: 📋 Plan eks-cluster
        run: terraform plan -out=tfplan

      - name: ✅ Apply eks-cluster
        run: terraform apply -auto-approve tfplan

  # deploy-app:
  deploy-app:
    name: 🚀 Deploy Web Application
    needs: deploy-eks
```



```
jobs:
  deploy-eks:
    steps:
      ...

  # deploy-app:
  deploy-app:
    name: 🚀 Deploy Web Application
    needs: deploy-eks
    runs-on: ubuntu-latest
    defaults:
      run:
        working-directory: WEB
    env:
      AWS_REGION: eu-west-2
      CLUSTER_NAME: laredo-cluster  # Update if different
      KUBECTL_VERSION: v1.30.1


    steps:
      - name: 📥 Checkout code
        uses: actions/checkout@v3

      - name: 🔐 Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v2
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: ${{ env.AWS_REGION }}

      - name: 📦 Install kubectl
        run: |
          KUBECTL_VERSION=v1.30.1  # or use: $(curl -s https://dl.k8s.io/release/stable.txt)
          echo "Installing kubectl version $KUBECTL_VERSION"
          curl -LO "https://dl.k8s.io/release/${KUBECTL_VERSION}/bin/linux/amd64/kubectl"
          chmod +x kubectl
          sudo mv kubectl /usr/local/bin/
```

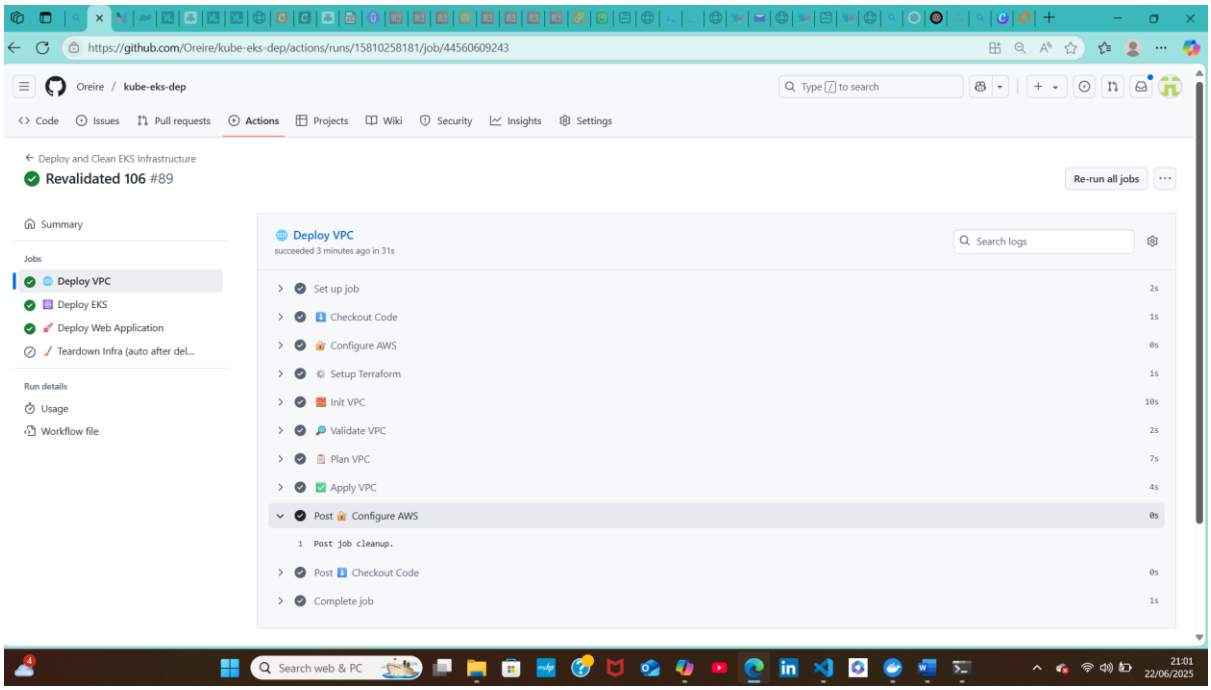


```
jobs:
  deploy-app:
    steps:
      - name: ✅ Confirm nodes are ready
        run: kubectl get nodes --no-headers | grep -c ' Ready'

      - name: 🚀 Apply Kubernetes manifests
        run: |
          kubectl apply -f web-deploy.yaml
          kubectl apply -f web-svc.yaml

      - name: ✅ Verify Deployment
        run: kubectl get deployment ghs-web -n coweb-ns

      - name: ✅ Verify Service Exposure
        run: kubectl get svc ghs-service -n coweb-ns

      - name: 🕐 Wait for LoadBalancer hostname
        run: |
          for i in {1..20}; do
            HOSTNAME=$(kubectl get svc ghs-service -n coweb-ns -o jsonpath="{.status.loadBalancer.ingress[0].hostname}")
            if [ -n "$HOSTNAME" ]; then
              echo "LoadBalancer endpoint is available at: http://$HOSTNAME"
              break
            fi
            echo "Waiting for LoadBalancer... ($i/20)"
            sleep 10
          done

      - name: 🌐 Print LoadBalancer endpoint
        run: |
          kubectl get svc ghs-service -n coweb-ns -o jsonpath="{.status.loadBalancer.ingress[0].hostname}"
          echo "LoadBalancer endpoint is available at: http://$(kubectl get svc ghs-service -n coweb-ns -o jsonpath="{.status.loadBalancer.ingress[0].hostname}")"

      - name: ✅ Verify rollout
        run: kubectl rollout status deployment/ghs-web -n coweb-ns
```

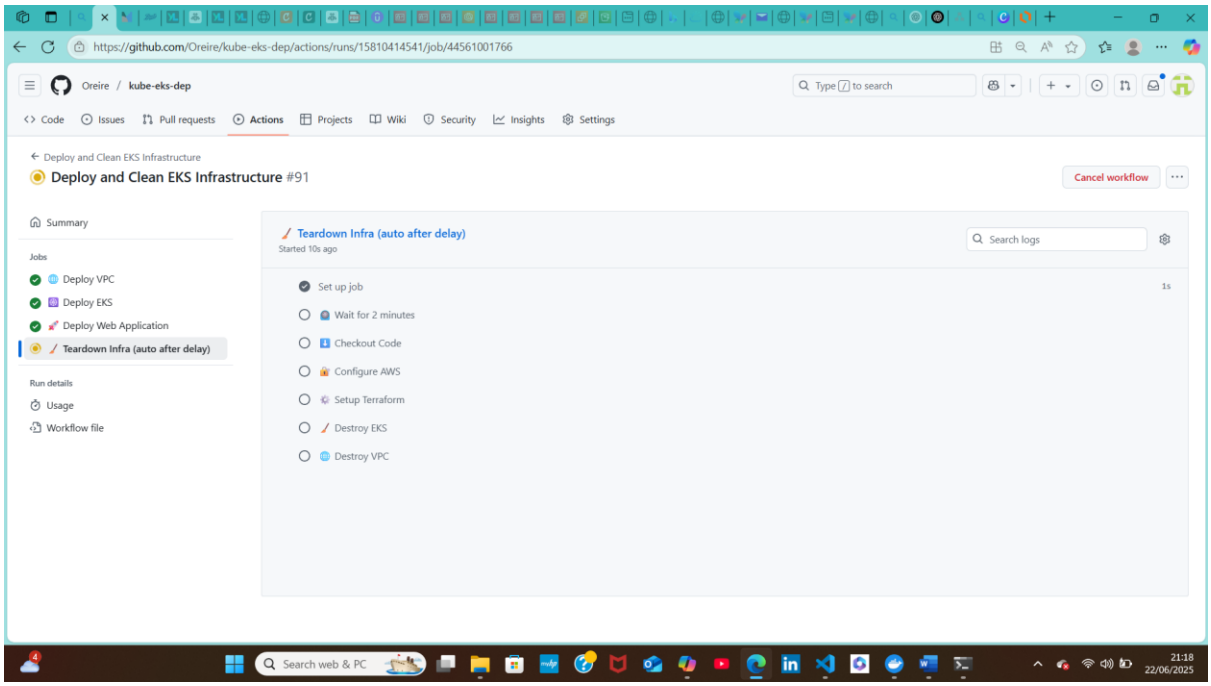


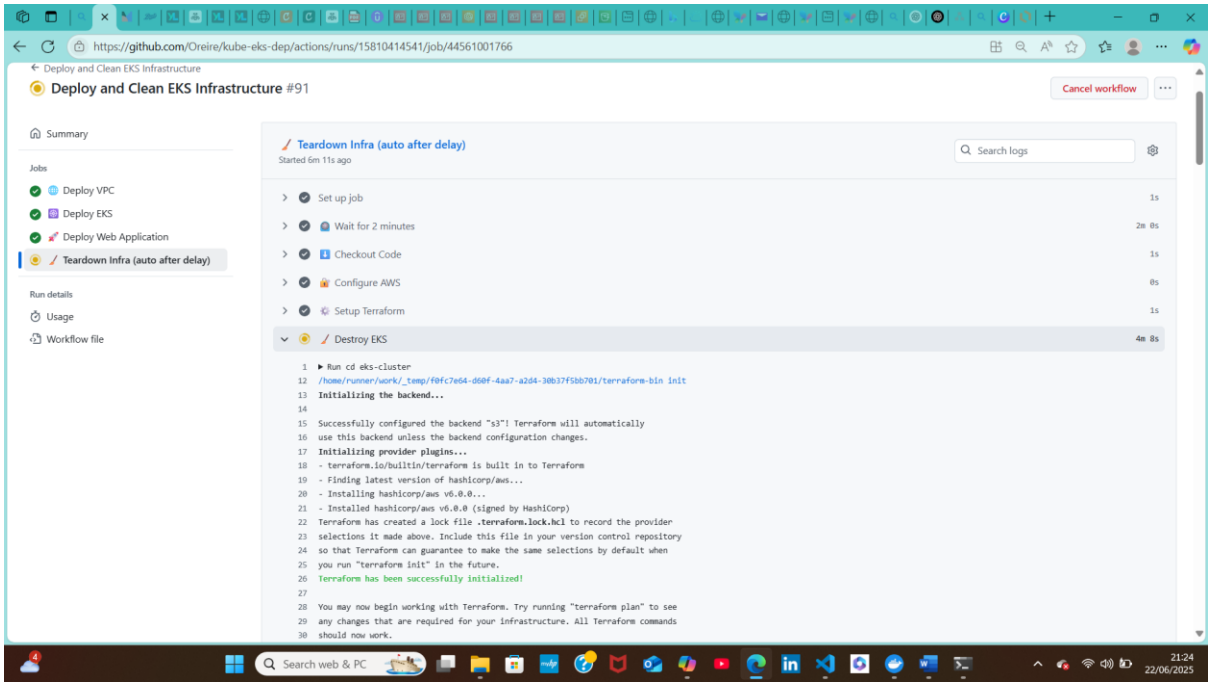
```
jobs:
  deploy-app:

  teardown-infra:
    if: github.event_name == 'workflow_dispatch' && github.event.inputs.auto_destroy == 'true'
    name: 🧹 Teardown Infra (auto after delay)
    needs: deploy-app
    runs-on: ubuntu-latest
    steps:
      - name: ⏳ Wait for 2 minutes
        run: sleep 120

      - name: ⬇️ Checkout Code
        uses: actions/checkout@v3

      - name: 🔐 Configure AWS
        uses: aws-actions/configure-aws-credentials@v2
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: eu-west-2

      - name: ⚙️ Setup Terraform
        uses: hashicorp/setup-terraform@v2

      - name: 🧹 Destroy EKS
        run: |
          cd eks-cluster
          terraform init
          terraform destroy -auto-approve

      - name: 🌐 Destroy VPC
        run: |
          cd ../KubeNet
          terraform init
          terraform destroy -auto-approve
```

GitHub Actions Pipeline



Teardown of Infrastructure





🛠 Conclusion

Deployed a production-grade EKS architecture with Terraform and GitHub Actions to provision and manage a resilient Kubernetes environment on AWS. The solution included modular Terraform layers, automated application deployment, and integrated teardown workflows for full infrastructure lifecycle management.

⚠ Key Challenges and Lessons Learned

Although the deployment of this production-grade EKS architecture was ultimately successful—thanks to thoughtful IAM planning, integrated teardown workflows, and a modular design—it required addressing several critical challenges along the way before achieving a truly stable, secure, and scalable infrastructure.

Below is a breakdown of key problem areas to watch for:

1. Network Complexity and Isolation

Managing the interplay between public and private subnets, NAT Gateways, security groups, and potential service mesh boundaries added significant complexity. Without precise configuration, the infrastructure risked misrouted traffic or unintended exposure of internal services.

2. IAM Role Misconfigurations

EKS heavily depends on tightly scoped IAM roles for control plane access, node group operations, and workload-level permissions. Seemingly minor missteps—such as omitting `eks:DescribeUpdate` or `iam:CreateRole`—caused deployment errors and blocked critical updates, prompting a revisit of role boundaries and trust relationships.

3. Bootstrap Failures in Node Groups

Some nodes initially failed to join the cluster due to incorrect IAM roles, improperly tagged subnets, or missing bootstrap configurations. Troubleshooting these issues emphasized the importance of validating node IAM roles, subnet reachability to the EKS endpoint, and proper user data scripts.

4. Terraform State Drift and Partial Teardowns

Orphaned infrastructure—such as lingering ENIs, EIPs, or route table associations—caused `DependencyViolation` errors during teardown. Ensuring logical deletion order, proper resource dependencies, and explicit lifecycle rules proved essential for clean and repeatable destruction.

5. Public LoadBalancer Exposure

While deploying services via an external LoadBalancer accelerated access, it also introduced potential security blind spots. To mitigate exposure risks, additional layers such as Kubernetes Network Policies, WAF integrations, and Ingress Controllers were identified as future hardening steps.

6. Update Collisions

Parallel update attempts on the EKS cluster often led to `ResourceInUseException` errors, particularly during overlapping Terraform apply events. Gating mechanisms, status polling, and sequential update strategies were introduced to prevent mid-flight configuration conflicts.

7. Observability Gaps

In the initial deployment, visibility into cluster health was limited. The absence of centralized logging and monitoring delayed response to performance issues. Integrating tools like Fluent Bit for logs and Prometheus/Grafana for metrics became a priority to reduce mean time to resolution (MTTR) and enhance operational insight.

☑ Outcome

Through systematic troubleshooting and iterative refinement of IAM roles, network topology, Terraform lifecycles, and deployment workflows, I delivered a stable, secure, and fully automated EKS architecture. The platform now supports GitOps-style delivery, modular provisioning, and clean teardown—positioned for further enhancements such as ArgoCD, secrets management, and workload observability.

Next steps:

🔐 1. Strengthening IAM Governance

- Refactor IAM roles using least-privilege principles with scoped permissions for provisioning, deployment, and teardown stages.
- Use tools like **AWS IAM Access Analyzer** or **Policy Sentry** to detect overly permissive policies.

🔍 2. Improved Observability

- Deploy **Fluent Bit** to forward cluster-wide logs to CloudWatch or an ELK stack.
- Set up **Prometheus** and **Grafana** for monitoring EKS health and application performance.
- Define SLOs for latency, availability, and error rates.

🧠 3. Enforcing CI/CD Validation

- Add pre-apply checks in GitHub Actions to validate:
    - Ongoing EKS updates
    - Terraform drift and plan output
- Integrate **OPA Gatekeeper** or **Polaris** for policy enforcement on manifests.

🔗 4. Adoption of ArgoCD for GitOps

- Transition from kubectl apply to **ArgoCD-managed deployments** using Kustomize overlays.
- Enable automated sync, pruning, and self-healing to maintain state consistency.

🔁 5. Improvements of Teardown Lifecycle

- Use tags or workspace identifiers to distinguish environments.
- Add pre-destroy checks for dangling resources like ENIs.
- Optionally implement TTL-based auto-cleanup using scheduled GitHub Actions.

🌐 6. Secured Network Boundaries

- Replacement of LoadBalancer services (AWS NLB@L3) with an **Ingress controller** (e.g., NGINX or AWS ALB) @ L7 of the OSI layer
- Enable Web Application Firewall (**WAF)** protection for public endpoints.
- Apply **Network Policies** to control inter-namespace traffic.

🔒 7. Centralization of Secrets Management

- Migrate to **AWS Secrets Manager**, **Parameter Store**, or **Sealed Secrets**.
- Enforce regular secret rotation and limit access via KMS policies and RBAC.